

User Manual

Contents

Welcome	1
Welcome to Webserver Stress Tool.....	1
Introduction: Testing Basics	2
Why testing?.....	2
The Business View.....	2
The Technical View.....	3
Performance, Load or Stress Testing?.....	4
Performance Tests.....	4
Load Tests.....	4
Stress Tests.....	4
Ramp Tests.....	5
Calculation of Load and Load Pattern.....	5
For Existing Websites.....	5
For New Websites.....	5
Playing With Numbers.....	5
When Should I Start Performance Testing?.....	6
Glossary.....	6
Webserver Stress Tool Features	7
Key Features.....	7
How much load can Webserver Stress Tool generate?.....	8
Webserver Stress Tool can be used for various tests.....	8
Testing Elements.....	8
Test results can be viewed as.....	9
Other Features.....	9
Installation	11
System-Requirements.....	11
Installation/Deinstallation.....	11
Configuring Webserver Stress Tool	12
Selecting the Test Type and the Number of Users.....	12
Test Type.....	13
User Simulation.....	13
Project/Scenario Comments, Operator.....	14
Selecting the URLs or Editing the URL Script.....	14
Using Simple URL Sequences.....	14
Choosing the URL Sequencing.....	16
Using the URL Recorder.....	16
Setting Up the Data Merging Feature.....	17
Tutorial for Data Merging.....	18

Using Custom URL Scripts for Advanced URL Sequences	19
OnBeforeClick.....	20
OnBeforeClick Samples	21
OnAfterClick	21
OnAfterClick Samples.....	22
OnBeforeRequest	22
OnBeforeRequest Samples	22
OnAfterRequest.....	22
OnAfterRequest Samples	23
Advanced URL Script Samples	23
Reading a TOKEN from a page and reusing it on subsequent requests	23
Load-Testing SOAP Servers	24
URL Script Function Reference	26
Global Variables.....	26
String Functions	27
Date/Time Functions	29
Arithmetic Functions	31
Filehandling Functions	31
Other Functions	31
Constants	32
Setting the Browser Simulation Parameters.....	32
Browser Simulation	33
Recursive Browsing.....	33
Setting Program Options.....	34
Advanced Settings	34
Logging	35
Local IP Addresses to use.....	35
Timer	35
Performance Tips&Tricks	36
Finding the Bottleneck of Your Test Setup	36
Network Issues.....	36
Test Client Issues.....	37
Running the Test	38
Reviewing Logfile Results	40
Summary Log and Detailed Log.....	40
User Logs.....	41
Results per User.....	41
Results per URL	42
Analyzing Graphical Results	44
Graph Basics	44
Usage of the Graphs.....	44
Hiding Graph Lines	44
Zooming/Panning Graphs.....	44
Graph's Context Menu	44
Graph Click Times & Errors (per URL)	45
Graph Click Times, Hits/s and Clicks/s	46
Graph Hierarchy	47
Graph Spectrum of Click Times	49
Graph Server and User Bandwidth	50

Graph Open Requests and Traffic.....	50
Graph Protocol Times.....	51
Graph Test Client's Health.....	52
Creating Reports	53
Report (Word).....	53
Report (HTML).....	54
Additional Features	56
Working with Different Test Scenarios.....	56
Command Line Interface.....	56
Running Several Tests at Once.....	56
Using Tokens.....	56
Tips and Tricks	58
Check out the Paessler Knowledge Base.....	58
Recording HTTP URLs for Complex Web Applications.....	58
Appendix	59
Script Syntax for URL Scripts.....	59
Basic syntax.....	59
Script structure.....	59
Identifiers.....	59
Assign statements.....	60
Character strings.....	60
Comments.....	60
Variables.....	60
Indexes.....	60
Arrays.....	61
if statements.....	61
while statements.....	61
loop statements.....	61
for statements.....	62
select case statements.....	62
function and sub declaration.....	63
Additional Functions.....	63
Useful RFCs.....	63
Useful Books.....	64
Andrew B. King: Speed Up Your Site.....	64
Splaine, Jaskiel: The Web Testing Handbook.....	65
Nguyen: Testing Applications on the Web.....	65
Nielsen: Designing Web Usability.....	65
Software License and Contact Information	66
Ordering Webserver Stress Tool.....	66
Copyright.....	66
Support.....	66
Consulting, Custom Versions of the Software.....	66
License/Usage Terms.....	66

Welcome

Welcome to Webserver Stress Tool

Most websites and web applications run smoothly and correctly as long as only one user (e.g. the original developer) or just a few users are visiting at a given time. But what happens if thousands of users access the website or web application at the same time?

Using Webserver Stress Tool you can simulate various load patterns for your webserver which will help you to find problems in your webserver set up. With steadily increasing loads (so called “ramp tests”) you are able to find out how much load your server can handle before serious problems arise.



Introduction: Testing Basics

Why testing?

The Business View

Many websites today have a serious business mission—to make money. And whether that's through providing custom content and proprietary services, through advertising opportunities or by selling retail products, these high-traffic websites and applications need to be up and running at all times. Because if performance slows even a little, fickle web users are likely to jump quickly to a competitor's site.

The message to website owners is clear: **Test and monitor your website!**

Few websites, if any, perform exhaustive testing. Usually focused solely on catching bugs, many websites ignore functionality testing, usability testing and performance testing—three critical elements in defining the user experience with a website or web application. In short, webmasters and developers should not only test for bugs, test whether the website does what it is meant to do (functionality testing) and test whether the user is able to easily accomplish tasks and objectives on the website (usability testing), but they must also test whether the user gets results from the website in an acceptable time (performance testing).

Performance testing is a critical component of your website or web application's overall success. From a performance standpoint, your goal is to ensure that your end-user's or customer's mouse click is not met with silence. Optimize your webserver so that that 95% of all web requests are processed in less than 10 seconds.

Jakob Nielsen, one of the foremost experts on software and website usability suggests the following performance thresholds for your website and or web application:

Download Time	User's View
< 0.1 s	User feels that the system is reacting instantaneously.
< 1.0 s	The user experience is not compromised. Although the user is unhappy with the wait, they are still focused on the current web page.
< 10 s	As wait times get close to 10s, studies

	have shown that the likelihood of user distraction increases greatly
> 10 s	User is most likely distracted from the current website and loses interest.

Webserver Stress Tool allows you quickly ascertain and identify performance problems so that you can quickly correct them to prevent user dissatisfaction and potential loss of revenue.

Through an intuitive interface, flexible testing parameters, and comprehensive reporting, Webserver Stress Tool provides you the tool to include performance testing as a regular part of website and web application maintenance and deployment.

Once your webserver has been deployed with the correct configurations (based upon performance testing), you may also consider deploying a 24/7 monitoring application. Paessler's IPCheck Server Monitor (<http://www.paessler.com>) can help you keep a constant, vigil eye on your investment in web architecture technology.

The Technical View

Although Webserver Stress Test 6 and performance testing in general solve key business issues such as up-time, user experience, and ROI, performance testing has a number of technical considerations to ensure that those business issues are resolved. For example, consider the following questions

- Is your webserver prepared for the traffic you are expecting?
- Is your webserver prepared for increasing visitors over the months and years to come?
- Can your webserver survive a massive spike in user traffic (e.g., if your website is mentioned on national TV or your company emails a newsletter to all customers and prospects)?
- How many users can your web server handle before users start getting error messages or server timeouts?
- How many seconds does it take for a visitor to your website to receive a page after clicking on a link? Under normal conditions? Under heavy conditions?
- Does your application or shopping cart support simultaneous users?
- Are your scripts and databases optimized to run as quickly as possible and do they interact with each other correctly under heavy webserver loads?
- Is the web hosting service doing a good job?
- Is your webserver's bandwidth sufficient?
- Is your server hardware sufficient?

Performance testing, as a valuable aspect of maintaining and growing the web portions of your business, is about answering these questions. To do an adequate job of representing your company to the world with your website, you need to discover the answers to all of these questions!

Performance, Load or Stress Testing?

Although many network technicians use these word synonymously, there are subtle but important differences.

Performance Tests

Performance tests are used to test each part of the webserver or the web application to discover how best to optimize them for increased web traffic. Most often this is done by testing various implementations of single web pages/scripts to check what version of the code is the fastest.

Webserver Stress Tool supports this type of test with the ability to run several (e.g. 20-100) simultaneous requests on one URL and record the average time to process those requests. By changing your website or application code under repeated tests, you can discover critical issues to address for optimal performance. Usually this type of test is run without requesting page images in order to concentrate the testing on the script and code itself.

Load Tests

Load tests are performed by testing the website using the best estimate of the traffic your website needs to support. Consider this a “real world test” of the website.

The first step is to define the maximum time it should take (from a usability and customer experience standpoint, not a technical one) for a page to load. Once you have determined this, you need to calculate the impact of exceeding that maximum time—will you lose sales? Will you lose prospective customers? A good rule of thumb is to make certain that no website visitor waits longer than ten (10) seconds for a web page to load.

Once this threshold has been determined, you have to calculate the anticipated load and load pattern for your website, which you can then simulate through Webserver Stress Tool. See the Calculation of Load and Load Pattern section for details on load and load pattern calculation.

At the end of the load test you can compare the test results with the your maximum request time threshold. When some page requests take longer than the target times or generate error messages, it is clear that there is work to be done to the application and webserver.

Stress Tests

Stress tests are simulated “brute force” attacks that apply excessive load to your webserver. “Real world” situations like this can be created by a massive spike of users –caused by a large referrer (imagine your website being mentioned on national TV...). Another example would be an email marketing campaign sent to prospective customers that asks them to come to the website to register for a service or request additional information. An inadvertent denial of service to prospects who are ready to learn more about your product could have a serious impact on your bottom line.

The purpose of a stress test is to estimate the maximum load that your webserver can support. Webserver Stress Tool can help you learn the traffic thresholds of your webserver and how it will respond after exceeding its threshold.

Ramp Tests

Ramp Tests are variations of Stress Tests in which the number of users is increased over the life of the test—from a single user to hundreds of users. By reviewing the graphs of click times and errors, a Ramp Tests can help you determine what maximum load a server can handle while providing optimal access to web resources

Calculation of Load and Load Pattern

Calculating the load and load pattern is probably the trickiest issue in conducting website performance tests.

First, remember that there is a difference between users, transactions, page views and hits:

- One user can conduct several transactions (e.g., visit a Homepage, search for a product, view a product's details, buy a product, etc.)
- One transaction can create several page views (e.g., add products to the shopping cart, go to the checkout, enter credit card, etc.)
- One page view can create multiple hits (e.g., framesets, images, applets, etc. for a single webpage)

For Existing Websites

If you already have your website online, a good way to start calculating the load and load pattern is to use a good log file analyzer on the log files produced by your webserver. Web log file analyzer tools will help you determine how many people access the site per day and per hour, what pages/scripts are used how often, etc. These logs will help you determine how many visitors and pageviews you have at specific times of the day as well as what your busiest day/time is and what pages are most popular.

For New Websites

If you are working on a new website, you have to ascertain load and load pattern yourself. One way to define the load pattern is:

- Step 1: Come up with the target number of users.
- Step 2: Define a couple of different “model users” (e.g. teenager, business professional, senior citizen, etc.) and surf from their point of view through the website. Track the web pages they access and gather these stats.

Playing With Numbers

At the end you should have a list of URLs and their frequency of use.

Try to answer the following question for each test scenario:

- How many users constitute a normal load? How many users constitute a peak load? How many, in each load, were simultaneous?
- How much time elapses between each user click?
- What URLs are visited the most?

- Are there any “paths” through the site? A path is defined as a per-defined or intuitive manner (through a specific sequence of URLs) to access resources on your site.

Remember to factor into your analysis that there could be spikes in your traffic (i.e., a holiday promotion or new advertising campaign).

Now feed this data into Webserver Stress Tool, hit “Start Test” and keep your fingers crossed!

When Should I Start Performance Testing?

The answer is simple: **You cannot start performance testing early enough when building web applications!**

For instance, it’s even a good idea to start performance testing before a single line of code is written. By testing the base technology (network, load balancer, application, database and web servers) early on for the load levels you plan to support, you can better optimize your webserver and potentially avert business costs (i.e., lost sales) later on. Discovering that your hardware configuration is inadequate when the application is deployed can be very expensive to correct. Testing the server for its maximum stress level before development begins is an excellent idea.

The costs for correcting a performance problem escalate as the development process moves forward. For instance, discovering a performance problem after an application or website is already deployed means countless man hours to correct the server issue—man hours that were already spent configuring the webserver (or application) the first time.

During software development, all software engineers (and the quality assurance team) should have access to performance test tools to test their own code for performance and for parallel execution problems (e.g., problems caused by database locks or other mutexes). Software engineering managers for web projects are realizing that each developer must be responsible for both the functionality and performance of code.

As soon as several web pages are working, the first load tests should be conducted by the quality assurance team. From that point forward, performance testing should be part of the regular testing routine each day for each build of the software.

Glossary

Here are some glossary terms used very often in the manual and inside the software:

- **Click**
A simulated mouse click of a user sending a request (one of the URLs from the URL list) to the server and immediately requesting any necessary redirects, frames and images (if enabled).
- **Request**
An HTTP request sent to the server regardless of an answer.
- **Hit**
A completed HTTP request (i.e. sent to the server and answered completely). Hits can be the PAGE request of a "click" or its frames, images etc.

- Time for DNS
Time to resolve a URL's domain name using the client system's current DNS server.
- Time to connect
Time to set up a connection to the server.
- Time to first byte (TFB)
Time between initiating a request and receiving the first byte of data from the server.
- Click Time
The time a user had to wait until his "click" was finished (including redirections/frames/images etc.).
- Click Delay
The time a user needs to view the webpage he just downloaded until he initiates the next click
- User Bandwidth
The bandwidth a user was able to achieve.
- Sent Requests
Number of requests sent to the server during a period.
- Received Requests
Number of answers received from the server during a period.

Webserver Stress Tool Features

Key Features

Webserver Stress Tool simulates anywhere from a few users to several hundred users accessing a website via HTTP/HTTPS at the same time.

Based on a set of URLs or using a VBScript the software simulates independent users requesting webpages from that URL including images, frames etc.

Each user is simulated by a separate thread with its own session information (e.g. cookies are stored individually for each user). URLs can be parameterized for each user and the sequence of URLs can be varied.

How much load can Webserver Stress Tool generate?

We have successfully tested Webserver Stress Tool 7 with

- more than ~500 MBit/s network load,
- more than ~1.000.000 pageviews per hour and
- up to 10.000 simultaneous users

but the actual load you can achieve is highly dependent on your network infrastructure, your server/client hardware, the file sizes and your web application.

Webserver Stress Tool can be used for various tests

- **Performance Tests** are used to test each part of the webserver or the web application to discover what parts, if any, are slow and how you can make them faster. Most often this is done by testing various implementations of single web pages/scripts to determine a configuration of code that is the fastest.
- **Load Tests** are performed by testing the website using the best estimate of the traffic your website must support. Consider this like a “real world” test of the website.
- **Stress Tests** are simulated “brute force” attacks that apply excessive load on your webserver. Real world situations like this can be created by a massive spike in users caused, innocently enough, by a new advertising campaign.
- **Ramp Tests** are used to determine the maximum threshold of users that can be served before error messages are produced.
- Other custom tests are also possible, e.g. tests to make sure that web pages can be requested simultaneously without problems, database deadlocks, semaphores etc.

Testing Elements

Webserver Stress Tool aggregates a number of different testing elements to help you get a holistic view of your entire website/webserver/application performance.

- **Click Time:** A simulated user’s mouse click that sends a request (one of the URLs from the URL list) to the server and immediately requesting any necessary redirects, frames and images (if enabled). The click time is calculated as the time between when the user clicked and when the server delivered the requested resources with all referenced items (images etc.).
- **Average Click Times:** The average values per URL, per user or per website
- **Time for DNS:** Time to resolve a URL's domain name using the client system's current DNS server.
- **Time to connect:** Time to set up a connection to the server.

- Time to first byte (TFB): Time between initiating a request and receiving the first byte of data from the server
- Request Time (TLB, Time to last Byte): Time for a single HTTP request (i.e. HTML page, image, frameset etc.)
- User/Server Bandwidth: The bandwidth a user and a server were able to achieve.
- Sent Requests: Number of requests sent to the server during a period.
- Received Requests: Number of answers received from the server during a period.
- Open Requests: Number of open request for a given moment
- Error rates: Number of failed request per time period, per user or per URL
- Webservice Stress Tool generates the applicable data elements for a specific test into a CSV-format log file for easy viewing.

Test results can be viewed as

Webservice Stress Tool also provides several ways to view results.

- Several easy to use graphs
- Summary Log
- Detailed Log
- User Log for each user
- Machine readable request log (CSV)
- Raw graph data (CSV)

Other Features

- Built-in report generator: Reports can be generated as HTML files and MS WORD documents
- Includes a URL recorder (complete web browser) to select the URL(s) you want to test (rather than typing them into a list)
- Works on any HTTP-URL or HTTPS-URL and can test any script (CGI, ASP, PHP etc.)
- Can also be used to test requests of larger download files (e.g. ZIP)
- Works with any webserver (no part of the software has to be installed on the server!)
- Includes support for
 - proxies (not for HTTPS) with optional proxy authentication
 - basic user authentication (username/password)
 - useragent string
 - any HTTP request header lines

- Individual cookie handling for each simulated user (e.g. ASP-Session-IDs)
- redirected requests and " http-meta-refresh " redirections
- several IPs for the client machine (up to 5000 IPs)
- data rate throttling (e.g. to simulate users accessing the server via a slow modem line)
- timeouts (e.g. to simulate surfers that click away after 20 seconds without answer of the server)
- When testing more than one URLs several URL selection methods can be selected to simulate different user behaviour
- Using a VBScript the URLs used for testing and various other parameters can be set individually
- Tests can run
 - until a specified number of clicks is reached for each user
 - until a specified time has passed
- Test can be started at a specified time
- Optional link checker can check all URLs for validity
- Test results can be stored into a ZIP for later reference

Installation

System-Requirements

The following Windows versions are supported:

- Windows 2000 (Workstation or Server)
- Windows XP (32bit and 64bit)
- Windows 2003 Server
- Windows Vista
- 32bit and 64bit versions are supported

Additionally you need a TCP/IP based network and a powerful test client machine.

Please also refer to the Performance Tips&Tricks Section!

Installation/Deinstallation

To install Webservice Stress Tool, run the **setup.exe** from the distribution **.ZIP** file. It is a common setup routine that should be completely self-explanatory.

To de-install the software at a later time, use the Add/Remove Software applet from Windows' Control Panel.

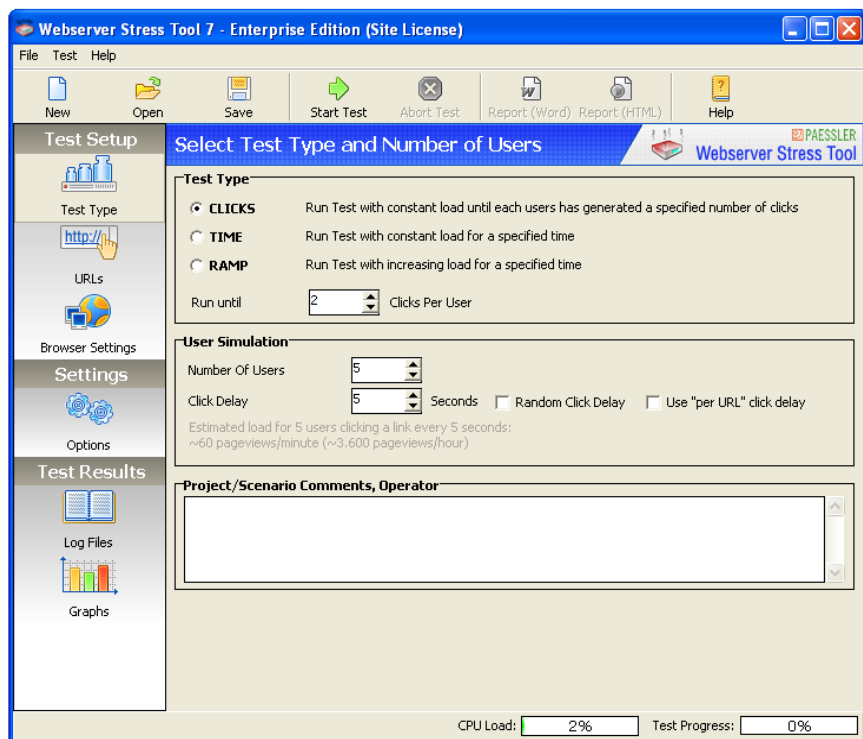
After Deinstallation please check the installation directory (usually c:\program files\Webserver Stress Tool) for any files that must be deleted manually. The de-installation process does not remove files that were created by the user (e.g., log files).

Configuring Webserver Stress Tool

Find the **Webserver Stress Tool** group in your Programs Menu and select **Webserver Stress Tool** to start the program.

Selecting the Test Type and the Number of Users

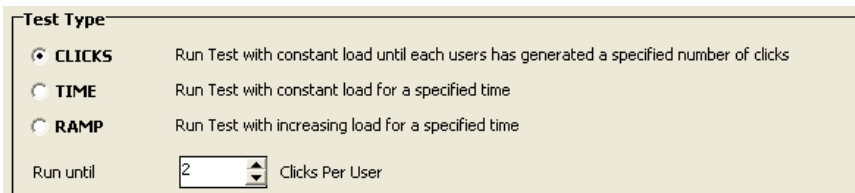
When you start Webserver Stress Tool for the first time, you automatically directed to the **Select Number of Users and Test Type** window. You can also click on “Test Type” in the left toolbar to access this panel:



This window allows you to enter the main test settings for the load pattern you want to simulate.

Test Type

Webserver Stress Tool offers three main test types



Test Type

CLICKS Run Test with constant load until each users has generated a specified number of clicks

TIME Run Test with constant load for a specified time

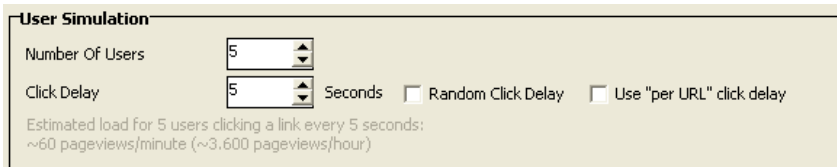
RAMP Run Test with increasing load for a specified time

Run until Clicks Per User

- **CLICKS:** the test is finished when each user has initiated the given number of clicks. CLICKS tests are the right choice to test specific URL sequences.
- **TIME:** tests that run for a specified number of minutes. A timed test is often used for “burn in tests”, e.g. to keep a server under full load for 10 hours.
- **RAMP:** Ramp tests also run for a specified time, but with increasing load from 1 user to the specified number of users which is reached at 80% of test time. During the last 20% the full number of users is active. A Ramp Test is a great way to find out the limitations of your webserver or web application.

User Simulation

Please enter the **Number Of Users** Webserver Stress Tool should simulate. This is the number of users that simultaneously use your website.



User Simulation

Number Of Users

Click Delay Seconds Random Click Delay Use "per URL" click delay

Estimated load for 5 users clicking a link every 5 seconds:
~60 pageviews/minute (~3,600 pageviews/hour)

The **Number of Users** can be a value between 1 and 10.000. But remember that the maximum number of simultaneous users that can be successfully simulated depends on the computing power of the client machine running Webserver Stress Tool and various parameters that you set later.

Webserver Stress Tool always shows the CPU load in the statusbar at the bottom and also generates a “client health” chart during the test. If you client machine runs at 100% CPU load you have hit your machine’s limit.

Next, you have to enter the **Click Delay** time for the simulated users. This setting is as important as the **Number of Users**. The lower the delay time between clicks, the greater the level of stress on your webserver.

Look at the “estimated load” calculation below the click delay setting to see what load your settings will create.

Important: These two values are the most critical values you will enter!

To create the highest possible load set the delay time between clicks to 0 (zero). This way Webserver Stress Tool will send the next user request immediately after the previous request is finished. Please note: When using the value of zero a setting of 40-80 users should be enough for most tests (higher values can decrease the load because of multithreading overhead).

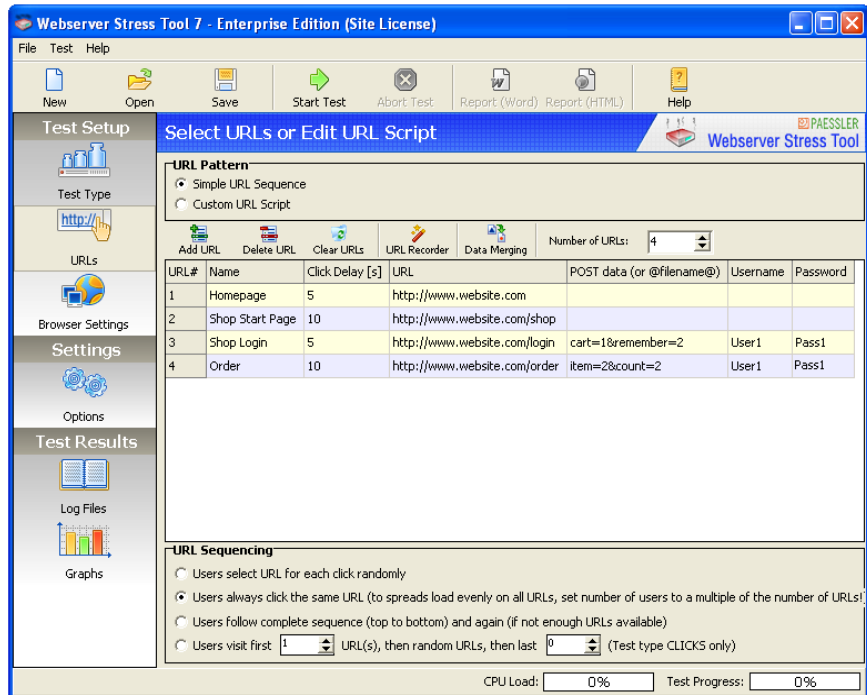
By enabling **Random Click Delay** you can tell Webserver Stress Tool to randomly use a delay time between two clicks that is between 0 seconds and the number of seconds you entered in **Click Delay**. This will make the test pattern even more dynamic but a little less reproducible.

Project/Scenario Comments, Operator

This is a great place to enter information about the test (i.e., parameters, reason for test, etc.). This comment will be inserted into test reports later and can help you to recreate the test later if necessary.

Selecting the URLs or Editing the URL Script

Click on the URLs button in the left hand toolbar to display the Select URLs Window:



You have two options to set the URLs for the test:

- **Simple URL Sequence:** For most simple tests you can simply enter your URLs here and choose an URL sequencing option
- **Custom URL Script:** For more complex tests you can also write a VB Script that selects the URLs and other parameters

Using Simple URL Sequences

Please enter your URLs (and – if needed – the other parameters for POSTDATA, usernames and password) into the table of URLs.

Add URL		Delete URL		Clear URLs		URL Recorder		Data Merging		Number of URLs: 4	
URL#	Name	Click Delay [s]	URL	POST data (or @filename@)	Username	Password					
1	Homepage	5	http://www.website.com								
2	Shop Start Page	10	http://www.website.com/shop								
3	Shop Login	5	http://www.website.com/login	cart=1&remember=2	User1	Pass1					
4	Order	10	http://www.website.com/order	item=2&count=2	User1	Pass1					

Here is a description of each field:

- **Name:** Select a descriptive Name for each URL entry. This name will be used in the graphs and in the logfiles (e.g. "Homepage", "Search", "Shopping Cart", "Checkout" etc.)
- **Click Delay:** Enter the time the simulated user shall take to "read the previous page". The simulated users will wait for this time (in seconds) after the previous URL has finished loading until this next URL will be clicked.
- **URL:** Enter the URL using the standard format **http://servername[:port]/path?get-params**. Here are some samples:
<http://www.server.com/home>
<http://www.server.com:8080/myfolder/myfile.php>
<http://www.server.com/signupform.cgi?username=name>
- **POSTDATA:** Usually Webserver Stress Tool creates GET requests. If you enter data into this column the request will be sent as POST request using the data that you provide (must be urlencoded). You can also use the content of a file for the POSTDATA by entering the filename with "@" at the beginning and at the end, e.g. **@mypostdata.txt@**. Note: This file must reside in the folder of the **webstress7.exe** file.
- **Username/Password:** If you use BASIC authentication (see Hypertext Transfer Protocol -- HTTP/1.0, <http://www.ietf.org/rfc/rfc1945.txt>, RFC 1945 for an explanation of BASIC authentication), enter the **Username** and **Password** for the URL here. With BASIC authentication, the login data is sent as part of the HTTP header in clear text. This will obviously not work for login mechanisms that use HTML FORMs. You have to simulate these logins using GET/FORM data. Note 1: NTLM or other authentications are not supported). Note 2: Don't mix up HTTP authentication and login mechanisms that use HTML forms.
- **Note:** from RFC 2617: "HTTP/1.0", includes the specification for a Basic Access Authentication scheme. This scheme is not considered to be a secure method of user authentication (unless used in conjunction with some external secure system such as SSL), as the user name and password are passed over the network as cleartext."

While editing the list you can use the following buttons:

- Click **Add URL** to add another line for a new URL at the bottom (you can also directly set the **Number of URLs** in the editbox). You can use up to 1000 URLs.
- Click **Delete URL** to delete the currently selected URL
- With **Clear URLs** you can clear the complete list

- The easiest way to get this list of URLs is to use the **URL recorder** (see below)
- Also the **Data Merging** feature is explained below.

Chossing the URL Sequencing

This setting determines how Webserver Stress Tool assigns the URLs to the users during the test.

URL Sequencing

Users select URL for each click randomly

Users always click the same URL (to spreads load evenly on all URLs, set number of users to a multiple of the number of URLs)

Users follow complete sequence (top to bottom) and again (if not enough URLs available)

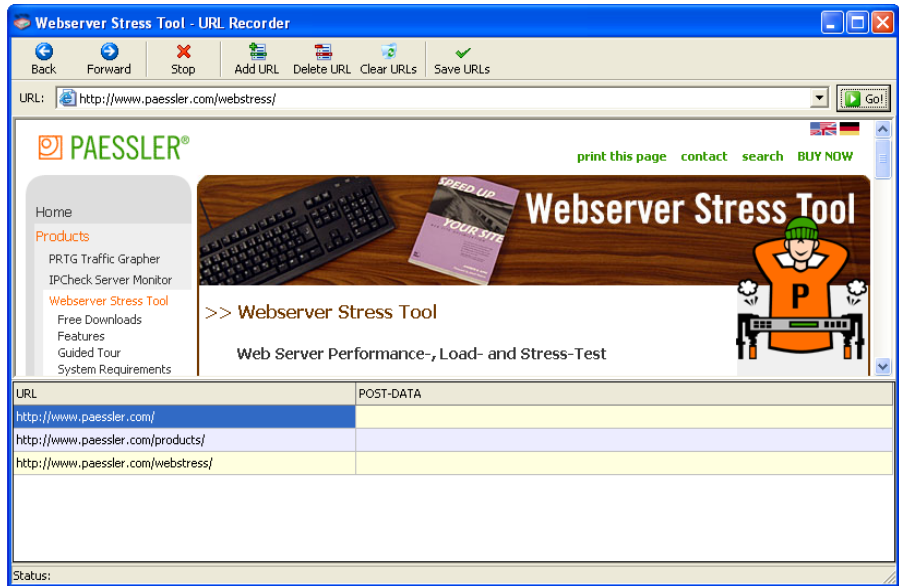
Users visit first URL(s), then random URLs, then last (Test type CLICKS only)

There are 4 options:

- **Users select URL for each click randomly:** Using the built in random function, Webserver Stress Tool simply selects one of the URLs for each click. Depending upon your website, this can be a good setting to create “real world” loads.
- **Users always click the same URL:** In the beginning of the test, each user selects a URL and clicks only this URL during the test. To spread the load evenly on all URLs, set the number of users to the number of URLs or a multiple of that. This setting is very useful in comparing the request times of different webpages (e.g. with different implementations of a feature) or to find out what pages are slower than others.
- **Users follow complete sequence:** All users will use URL#1 for the first click, URL#2 for the second click and so on. If a user reaches the last URL he will start with URL#1 again. Use this setting to simulate paths through your website, e.g. to put products into and order from a shopping cart.
- **Users visit first X URLs, then random, then last X URLs:** All users will use the first X URLs (top to bottom). After that, the remaining URLs are assigned randomly. For CLICKS tests you have the additional option to set a number of URLs at the bottom of the URL list the users should visit at the end of the test. This would be an appropriate test pattern if you have a website in which users have to login using a couple of URLs, then surf around and log out at the end.

Using the URL Recorder

Webserver Stress Tool offers a “click recorder” to build the list of test URLs. Click on **URL Recorder** to start the click recorder:



Simply enter the first URL in the **URL** edit field and click on **Go!**. Watch the list of URLs in the lower part of the window. Every time you click a link on the browser window or submit a FORM, the URL is appended to the list.

If a POST request is submitted, the POST data is also saved. If a page is a frameset, all URLs of the frameset are added to the list.

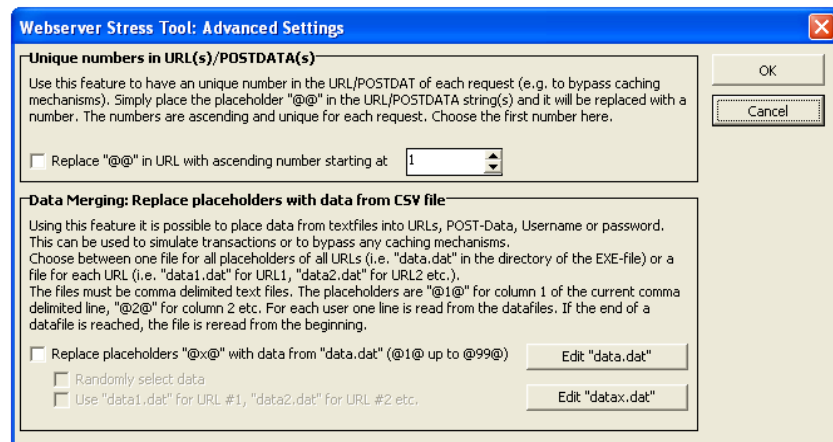
If a click opens a new window, Webservers Stress Tool will also open a new window and record further clicks.

You may edit the list of URLs using **Add URL**, **Delete URL** and **Clear URLs**.

Please note: In the upper part of the window the "Internet Explorer OCX Control" is called as the web browser, which uses the currently installed version of IE on your system.

Setting Up the Data Merging Feature

To be more flexible with the URLs or to bypass caching mechanisms (e.g. of a web-, application- or proxy-server in your test setup), Webservers Stress Tool offers the possibility to merge additional dynamic data into the requested URLs.



Unique numbers in URL(s): Use this feature to have a unique number in the URL of each request (e.g. "cachebuster=@@" to bypass caching mechanisms).

Simply place the placeholder "@@" in the URL string(s) and it will be replaced with a number. The numbers are ascending and unique for all users and clicks.

Replace placeholders with data from file: Use this feature to place data from text files into URLs, POST-Data, Username or password. This can be used to simulate transactions or to bypass caching mechanisms.

Choose between one file for all placeholders of all URLs (i.e. "data.dat" in the directory of the EXE-file) or a file for each URL (i.e. "data1.dat" for URL1, "data2.dat" for URL2 etc.).

You can edit the files by clicking **Edit "data.dat"** and **Edit "datax.dat"** respectively.

If you need to edit the files manually, please keep in mind the following. Files must be comma delimited text files. The placeholders are "@1@" for column 1 of the current comma delimited line, "@2@" for column 2 etc. For each user one line is read from the data files. If the end of a data file is reached, the file is reread from the beginning.

Note: if there are spaces in a column please use quotation marks around the data and double quotation marks for a single quotation mark, e.g.

1,"one and two",3,4,"four and five","This "" is a quotation mark"

Tutorial for Data Merging

Let's assume you have three URLs a user is required to go through to login into your site and do something.

- <http://myserver/homepage.htm> (standard homepage with a login form)
- <http://myserver/login.htm> (POST parameters are login/password, this page sets a cookie)
- <http://myserver/data.htm> (some GET parameters)

You want to simulate 10 different users logging into the site and going through these 3 urls.

Using Webserver Stress Tool this can be done as follows:

- On page **Test Setup**.
- Select **Test Type CLICKS**, set **number of clicks** to 3, set **Number of Users** to 10
- On page **URLs**
- Set **Number of URLs** to 3, select **All Users follow complete Sequence**, Enter the three URLs
- For URL#2 enter the following in the POSTDATA column:
username=@1@&password=@2@
- For URL#3 change the URL like this:
http://myserver/data.htm?data=@1@
- Click on **Data Merging**
- Enable **Replace Placeholders...** and enable **Use data1.dat for URL#1...**

- Click on **Edit "Datax.dat"**
- Answer "**Edit data for what URL**" with the value 2
- enter the username and password combinations into column **@1@** and **@2@** and click **OK**
- Again Click on **Edit "Datax.dat"**
- Answer "**Edit data for what URL**" with the value 3
- Enter the data for URL#3 into column **@1@** and click **OK**
- Click OK again
- Review all the other settings
- Run your test

At first, all 10 users send a request for URL#1 which is the plain homepage.

Then each user requests the login.htm URL with but the **@1@** and **@2@** in the POSTDATA field replaced by values from the file data2.dat so that every request is sent with different login data (**@1@** is replaced with data from column 1, **@2@** for column 2).

If login.html sends a cookie, this cookie is stored individually for each user and is resent with the third request..

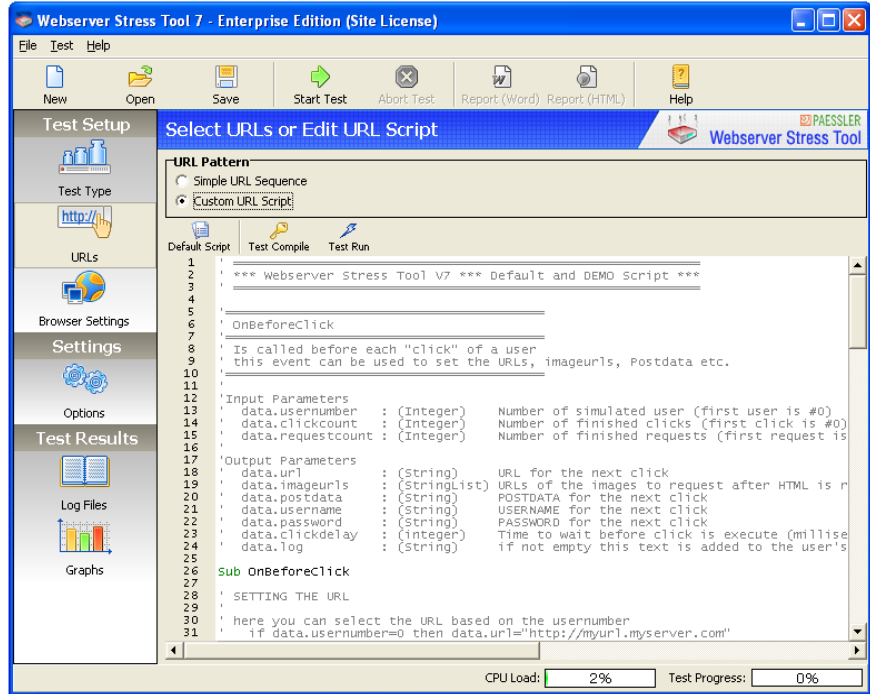
For the third click Webserver Stress Tool replaces the **@1@** placeholder with the strings from data3.dat, thus sending 10 different GET urls to the server along with the cookies received in run 2.

You can examine the log files to make sure that the data was sent in the way you expected.

Using Custom URL Scripts for Advanced URL Sequences

If simple URL sequences are not flexible enough for your testing needs then you should consider using Custom URL Scripts to configure Webserver Stress Tool.

On the **URLs tab** choose **Custom URL Script** to enable this feature. A code editor comes up where you can edit your URL Script:



A good start is to click Default Script, then Webservers Stress Tool loads the built in demo script.

The script language is very similar to VBScript and description of the syntax and a list of allowed commands can be found in the Appendix.

The script must define four main functions: **OnBeforeClick**, **OnAfterClick**, **OnBeforeRequest** and **OnAfterRequest**.

OnBeforeClick

OnBeforeClick is called before each "click" of a user. This event can be used to set the URLs, imageurls, Postdata etc.

Input Parameters:

data.usernumber	(Integer)	Number of simulated user (first user is #0)
data.clickcount	(Integer)	Number of finished clicks (first click is #0)
data.requestcount	(Integer)	Number of finished requests (first request is #0)
data.token	(String)	Use this variable to store e.g. tokens from one click or request to the next (see Advanced Samples section below)

Output Parameters:

data.url	(String)	URL for the next click
data.addimageurl	(String)	Adds the given URL to the list of image URLs that are requested after HTML is received
data.postdata	(String)	POSTDATA for the next click
data.username	(String)	USERNAME for the next click

data.password	(String)	PASSWORD for the next click
data.clickdelay	(integer)	Time to wait before click is execute (milliseconds)
data.log	(String)	if not empty this text is added to the user's log
data.soapaction	(String)	If you want to test a SOAP server set this variable to the string of your SOAPAction (see Advanced Samples Section below)

OnBeforeClick Samples

Selecting the URL based on the usernumber

```
if data.usernumber=0 then data.url="http://myurl" end if
if data.usernumber=1 then data.url="http://myurl2" end if
if data.usernumber=2 then data.url="http://myurl3" end if
```

Selecting the URL based on the clicknumber

```
if data.clickcount=0 then data.url="http://myurl" end if
if data.clickcount=1 then data.url="http://myurl2" end if
if data.clickcount=2 then data.url="http://myurl3" end if
```

Setting the image URLs from the script (instead of using the "download images/frames etc" feature of WebStress, which is very CPU cycle consuming, you can tell Webserver Stress Tool the URLs it should request after requesting the main HTML). AddimageURL adds each assigned URL to the list of image URLs.

```
data.addimageurl="http://my.server.com/image1.gif"
data.addimageurl="http://my.server.com/image2.gif"
data.addimageurl="http://my.server.com/image3.gif"
```

Setting the Click Delay (you can set the time before this user initiates his mouseclick (in milliseconds), e.g. using a random value)

```
data.clickdelay=random*(10000+data.usernumber*40)
```

Setting POSTDATA and credentials

```
data.postdata="MyPostData"
data.username="username"
data.password="password"
```

Writing to the user's logfile

```
data.log="Preparing click #"+inttostr(data.clickcount+1)+" of user
#"+inttostr(data.usernumber+1)
```

Reading the POSTDATA from a file (please edit the filename/filepath for your needs):

```
data.postdata=loadstringfromfile("D:\temp\mypostdata")
```

OnAfterClick

OnAfterClick is called after each "click" of a user and can be used to do some extended logging or to analyze the HTML code.

Input Parameters:

data.usernumber	(Integer)	Number of simulated user (first user is #0)
data.clickcount	(Integer)	Number of finished clicks (first click is #0)
data.requestcount	(Integer)	Number of finished requests (first request is #0)

Output Parameters:

data.log (String) if not empty this text is added to the user's log

OnAfterClick Samples

Writing to the user's logfile

```
data.log="Finished click #"+inttostr(data.clickcount+1)+" of user  
#"+inttostr(data.usernumber+1)
```

OnBeforeRequest

OnBeforeRequest is called before each single HTTP Request of a user (i.e. clicks, images, frames etc.) and can be used to log data or to alter the HTTP headers.

Input Parameters:

data.usernumber	(Integer)	Number of simulated user (first user is #0)
data.clickcount	(Integer)	Number of finished clicks (first click is #0)
data.requestcount	(Integer)	Number of finished requests (first request is #0)
data.requesttype	(String)	Type of Request (e.g. CLICK, IMAGE, FRAME)

Output Parameters:

data.log	(String)	if not empty this text is added to the user's log
request.additionalheader	(String)	Additional lines for the HTTP header sent to the server

OnBeforeRequest Samples

Writing to the user's logfile

```
data.log="Doing a "+data.requesttype+"-request for click  
#"+inttostr(data.clickcount+1)+" of user #"+inttostr(data.usernumber+1)
```

Adding custom text to the HTTP Header

```
request.additionalheader="MyOwnHeaderline"
```

OnAfterRequest

OnAfterRequest is called after each single HTTP Request of a user (i.e. clicks, images, frames etc.) and can be used to log data and parse the results. E.g. if you need some part of the HTML code to be reused in subsequent requests this is the place to extract this string from the HTML.

Input Parameters:

data.usernumber	(Integer)	Number of simulated user (first user is #0)
data.clickcount	(Integer)	Number of finished clicks (first click is #0)
data.requestcount	(Integer)	Number of finished requests (first request is #0)
data.requesttype	(String)	Type of Request (e.g. CLICK, IMAGE, FRAME)

request.html	(String)	Resulting HTML of this request (can be raw GIF/JPG data for images)
request.receivedheader	(String)	Resulting HTTP Headers from the server
request.result	(String)	Result of a request (e.g. OK, Error)
request.resultcode	(String)	HTTP-Resultcode of a request (e.g. 200, 404, etc)

Output Parameters:

data.log	(String)	if not empty this text is added to the user's log
data.token	(String)	Use this variable to store e.g. tokens from one click or request to the next
request.additionalheader	(String)	Additional lines for the HTTP header sent to the server

OnAfterRequest Samples

Writing to the user's logfile

```
data.log="Finished request number "+inttostr(data.requestcount)+" with
resultcode "+inttostr(request.resultcode)+" (" +request.result+)"
```

The following code dumps HTML and headers into the log

```
data.log=data.log+crlf+"==header====="+c
rlf+request.receivedheader+crlf+"====="
====="
data.log=data.log+crlf+"==html====="+c
rlf+request.html+crlf+"====="
```

Writing the HTML (or any other data) of a request to a diskfile. Please edit the filename/filepath for your needs!

```
a=savestringtofile("d:\temp\Data of user number "
+inttostr(data.usernumber)+" request number "
+inttostr(data.requestcount)+".txt",request.html)
if a<>0 then data.log="Could not write file (result="
+inttostr(a)+)" end if
```

Advanced URL Script Samples

Reading a TOKEN from a page and reusing it on subsequent requests

The following script shows how to read some data from the HTML of a page and the use this data in subsequent requests:

```

' =====
' Webserver Stress Tool V7 *** Sample Script for Reusing a Token
' =====
' Requires Webserver Stress Tool 7.0.2 or later

Sub OnBeforeClick

    if data.clickcount=0 then data.url="http://walldorf.paessler.com"
end if
    if data.clickcount=1 then
data.url="http://walldorf.paessler.com/?test="+data.token end if
    data.log="Preparing click #"+inttostr(data.clickcount+1)+" of user
    #"+inttostr(data.usernumber+1)

end Sub

Sub OnAfterClick
    data.log=""
end sub

Sub OnbeforeRequest
    data.log=""
end sub

Sub OnAfterRequest

    if data.clickcount=0 then 'we only look in the HTML of the first
click for our tags
        if data.requestcount=0 then 'we only look in the HTML of the first
request of the first click for our tags

            tagbefore="<title>"
            tagafter="</title>"    'our tag/token delimiters

            if pos(tagbefore,request.html)>0 then
                if pos(tagafter,request.html)>0 then
                    tagbegin=pos(tagbefore,request.html)+length(tagbefore)
                    taglength=pos(tagafter,request.html)-tagbegin
                    mytag=copy(request.html,tagbegin,taglength)
                    data.log="FOUND TOKEN: '"+mytag+'"'
                    data.token=mytag
                else
                    data.log="Closing Token not found ("&tagafter&") in request
number "+inttostr(data.requestcount)+" with resultcode
"+inttostr(request.resultcode)+" ("&request.result&")"
                end if
            else
                data.log="Opening Token not found ("&tagbefore&") in request
number "+inttostr(data.requestcount)+" with resultcode
"+inttostr(request.resultcode)+" ("&request.result&")"
            end if

        end if
    end if

end sub

```

Load-Testing SOAP Servers

With Webserver Stress Tool you can perform load and stress tests for SOAP Servers/SOAP Services. SOAP method calls are nothing else than HTTP requests that send an XML dataset using a POST request to a webserver and then receive the results as an XML string.

Even though Webserver Stress Tool is not specialized in reading and writing the XML data for these requests, you can still use it as a load generator for your SOAP services.

This sample shows how to use Webserver Stress Tool. to issue a SOAP request to Google's Webservices API. Note: Of course you should refrain from load testing Google's webservers!

The trickiest thing of course is to find out the three input parameters. You must get this information from the SOAP server's documentation. For Google these are:

HTTP URL <http://api.google.com/search/beta2>

SOAPACTION "urn:GoogleSearchAction"

The XML-POSTDATA for the request should be stored into a file on your disk. To run a search request on Google the XML would be:

```
<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="urn:GoogleSearch"
xmlns:types="urn:GoogleSearch/encodedTypes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <tns:doGoogleSearch>
      <key xsi:type="xsd:string">*** PLACE YOUR GOOGLE API ACCESS KEY
HERE ***</key>
      <q xsi:type="xsd:string">paessler</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      <filter xsi:type="xsd:boolean">>false</filter>
      <restrict xsi:type="xsd:string" />
      <safeSearch xsi:type="xsd:boolean">>false</safeSearch>
      <lr xsi:type="xsd:string" />
      <ie xsi:type="xsd:string" />
      <oe xsi:type="xsd:string" />
    </tns:doGoogleSearch>
  </soap:Body>
</soap:Envelope>
```

Having this information we can now set up the URL script for Webserver Stress Tool for our test. By setting a value for data.soapaction we instruct Webserver Stress Tool to actually send a SOAP request with content type "text/xml":

```
Sub OnBeforeClick
  data.url="http://api.google.com/search/beta2"
  data.postdata=loadstringfromfile("c:yourpath\soaprequest.txt")
  data.soapaction="urn:GoogleSearchAction"
end Sub

Sub OnAfterClick
  data.log=" "
end sub

Sub OnbeforeRequest
  data.log=" "
end sub

Sub OnAfterRequest
  data.log=" "
end sub
```

After running the test the results from Google can then be reviewed if you enable "Save HTML to files"

```

xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="ns1:GoogleSearchResult">
<directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]">
</directoryCategories>
<documentFiltering xsi:type="xsd:boolean">>false</documentFiltering>
<endIndex xsi:type="xsd:int">10</endIndex>
<estimateIsExact xsi:type="xsd:boolean">>false</estimateIsExact>
<estimatedTotalResultsCount
xsi:type="xsd:int">28000</estimatedTotalResultsCount>
<resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[10]">
<item xsi:type="ns1:ResultElement">
<URL xsi:type="xsd:string">http://www.paessler.com/</URL>
<cachedSize xsi:type="xsd:string">11k</cachedSize>
<directoryCategory xsi:type="ns1:DirectoryCategory">
<fullViewableName
xsi:type="xsd:string">Top/Computers/Software/Internet/site_management/Monitorin
g</fullViewableName>
<specialEncoding xsi:type="xsd:string"></specialEncoding>
</directoryCategory>
<directoryTitle
xsi:type="xsd:string">&lt;b&gt;Paessler&lt;/b&gt;</directoryTitle>
<hostName xsi:type="xsd:string"></hostName>
<relatedInformationPresent
xsi:type="xsd:boolean">>true</relatedInformationPresent>
<snippet xsi:type="xsd:string">Powerful, easy-to-use software solutions from
&lt;b&gt;Paessler&lt;/b&gt;. Network management with&lt;br&gt; PRTG. Network
monitoring with IPCheck Server Monitor.</snippet>
<summary xsi:type="xsd:string">Provides software to test and monitor

```

Etc.

URL Script Function Reference

Global Variables

In order to exchange data between the users' threads (e.g. for global counters) use these global variables:

Global Variables:

global.integer1	(Integer)	Free usable global integer value
global.integer2	(Integer)	Free usable global integer value
global.integer3	(Integer)	Free usable global integer value
global.integer4	(Integer)	Free usable global integer value
global.integer5	(Integer)	Free usable global integer value
global.string1	(String)	Free usable global string value
global.string2	(String)	Free usable global string value
global.string3	(String)	Free usable global string value
global.string4	(String)	Free usable global string value
global.string5	(String)	Free usable global string value
global.float1	(Float)	Free usable global float/date value
global.float2	(Float)	Free usable global float/date value
global.float3	(Float)	Free usable global float/date value
global.float4	(Float)	Free usable global float/date value
global.float5	(Float)	Free usable global float/date value

Samples:

```
global.integer1=global.integer1+1
global.string1=global.string1+" MORE"
global.float1=now
data.log="counter="+inttostr(global.integer1)+" "+global.string1+"
time="+timetostr(global.float1)
```

String Functions

Copy(S; Index, Count: Integer): string

Copy returns a substring containing Count characters or elements starting at S[Index].

```
s=Copy("testtext",1,4)
```

Delete(var S: string; Index, Count:Integer)

Delete removes a substring of Count characters from string S starting with S[Index].

```
a="testtexttext"
Delete(a,4,4)
```

Insert(Source: string; var S: string; Index: Integer)

Insert merges Source into S at the position S[index].

```
a="testtexttext"
Insert("text",a,5)
```

Pos(Substr: string; S: string): Integer

Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions. Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero.

```
a=pos("sub","textsubtest")
```

Length(a:string):integer

Length returns the number of characters actually used in the string or the number of elements in the array.

```
a=length("teststring")
```

UpperCase(s:string) :string

UpperCase returns a copy of the string S, with the same text but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase. To convert 8-bit international characters, use AnsiUpperCase instead.

```
a=UpperCase("Test")
```

LowerCase(s:string):string

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase.

```
a=LowerCase("Test")
```

CompareStr(s1,s2:string):integer

CompareStr compares S1 to S2, with case-sensitivity. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2. The compare operation is based on the 8-bit ordinal value of each character and is not affected by the current locale.

```
b=CompareStr("Test","test")
```

CompareText(s1,s2:string):integer

CompareText compares S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. CompareText is not case sensitive and is not affected by the current locale.

```
b=CompareText("Test","test")
```

AnsiUpperCase(s:string):string

AnsiUpperCase returns a string that is a copy of S, converted to upper case. The conversion uses the current locale. This function supports multi-byte character sets (MBCS).

```
a=AnsiUpperCase("Test")
```

AnsiLowerCase(s:string):string

AnsiLowerCase returns a string that is a copy of the given string converted to lower case. The conversion uses the current locale. This function supports multi-byte character sets (MBCS).

```
a=AnsiLowerCase("Test")
```

AnsiCompareStr(s1,s2:string):integer

AnsiCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2.

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. Thus, setting S1 to 'a' and S2 to 'A' causes AnsiCompareStr to return a value less than zero, while CompareStr, with the same arguments, returns a value greater than zero.

```
b=AnsiCompareStr("Test","test")
```

AnsiCompareText(s1,s2:string):integer

AnsiCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. AnsiCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

```
b=AnsiCompareText("Test","test")
```

Trim(s:string):string

Trim removes leading and trailing spaces and control characters from the given string S.

```
a=Trim(" Test ")
```

TrimLeft(s:string):string

TrimLeft returns a copy of the string S with leading spaces and control characters removed.

```
a=TrimLeft(" Test ")
```

TrimRight(s:string):string

TrimRight returns a copy of the string S with trailing spaces and control characters removed.

```
a=TrimLeft(" Test ")
```

IntToStr(a:integer):string

IntToStr converts an integer into a string containing the decimal representation of that number.

```
b=IntToStr(12)
```

IntToHex(value:integer;digits:integer):string

IntToHex converts a number into a string containing the number's hexadecimal (base 16) representation. Value is the number to convert. Digits indicates the minimum number of hexadecimal digits to return.

```
a=IntToHex(12,4)
```

StrToInt(s:string):integer

StrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number.

```
a=StrToInt("12")
```

StrToIntDef(s:string;default:integer):integer

StrToIntDef converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToIntDef returns Default.

```
a=StrToIntDef("12",1)
```

FloatToStr(a:float):string

FloatToStr converts the floating-point value given by Value to its string representation. The conversion uses general number format with 15 significant digits.

```
s=floattostr(1.234)
```

Date/Time Functions

The script language uses the following definition for date and time values: The integral part of a value is the number of days that have passed since 12/30/1899. The fractional part of a value is fraction of a 24 hour day that has elapsed.

Following are some examples of TDateTime values and their corresponding dates and times:

0 12/30/1899 12:00 am

2.75 1/1/1900 6:00 pm

-1.25 12/29/1899 6:00 am

35065 1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value.

EncodeDate(Year, Month, Day: Word): DateTime

Returns a TDateTime value from the values specified as the Year, Month, and Day parameters. The year must be between 1 and 9999. Valid Month values are 1 through 12. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value.

For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

```
d=EncodeDate(2005,6,5)
```

EncodeTime(Hour, Min, Sec, MSec: Word): DateTime

Encodes the given hour, minute, second, and millisecond into a DateTime value. Valid Hour values are 0 through 23. Valid Min and Sec values are 0 through 59. Valid MSec values are 0 through 999. The resulting value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

```
d=EncodeTime(19,5,4,200)
```

DecodeDate(Date: DateTime; var Year, Month, Day: integer)

Breaks the value specified as the Date parameter into Year, Month, and Day values.

```
y=0  
m=0  
d=0  
DecodeDate(35065,y,m,d)
```

DecodeTime(Time: DateTime; var Hour, Min, Sec, MSec: Word)

DecodeTime breaks the object specified as the Time parameter into hours, minutes, seconds, and milliseconds.

```
h=0  
m=0  
s=0  
ms=0  
DecodeTime(1.978,h,m,s,ms)
```

DayOfWeek(Date: TDateTime): Integer

Returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

```
a=DayOfWeek(35065)
```

Date: DateTime

Use Date to obtain the current local date as a TDateTime value. The time portion of the value is 0 (midnight).

```
d=date
```

Now: DateTime

Returns the current date and time, corresponding to the sum of the value returned by the global Date and Time functions. Now is accurate only to the nearest second.

```
t=now
```

DateToStr(Date: TDateTime): string

Use DateToStr to obtain a string representation of a date value that can be used for display purposes.

```
DateToStr(35065.3455)
```

TimeToStr(Date: TDateTime): string

Use TimeToStr to obtain a string representation of a time value that can be used for display purposes.

```
TimeToStr(2445.3455)
```

DateTimeToStr(Date: TDateTime): string

Use DateTimeToStr to obtain a string representation of a date and time value that can be used for display purposes.

```
DateTimeToStr(35065.3455)
```

Arithmetic Functions

Round(a:float):integer

Round function rounds a real-type value to an integer-type value.

```
a=Round(12.5)
```

Trunc(a:float):integer

the Trunc function truncates a real-type value to an integer-type value.

```
a=Trunc(12.5)
```

Dec(a:integer or float)

Dec subtracts one from a variable.

```
Dec(a)
```

Inc(a:integer or float)

Inc adds one to the variable.

```
Inc(a)
```

Random

Random returns a random number within the range $0 \leq X < 1$.

```
A=random(10)
```

Filehandling Functions

LoadStringFromFile(filename:string):string

Loads a file into a string.

```
S=loadstringfromfile("c:\yourpath\myfile.txt")
```

SaveStringToFile(filename:string)

Saves a string into a file.

```
savestringtofile (s,"c:\yourpath\myfile.txt")
```

Other Functions

Beep

Beep generates a message beep

Beep

Constants

crlf

Returns a line break string (ASCII characters 13 and 10)

S=crlf

quotechar

Returns a quote character “

S=quotechar

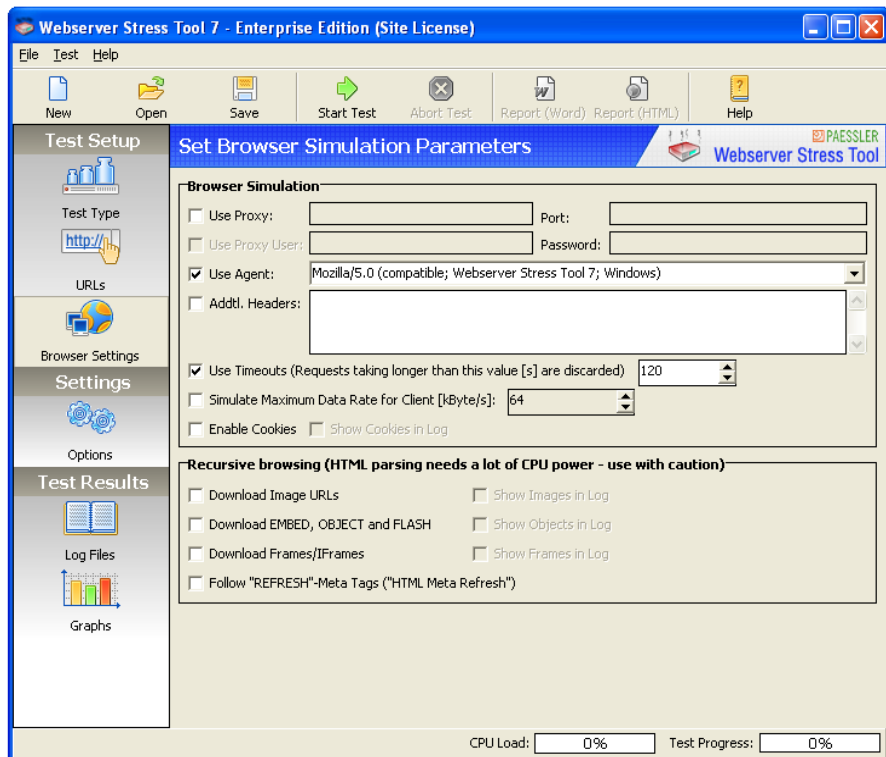
colonchar

Returns a colon character ‘

S=colonchar

Setting the Browser Simulation Parameters

Many characteristics of the simulated browser can be set by the user:



First, you need to note that Webrserver Stress Tool simulates a browser only from a “server’s point of view” (e.g., sending the requests, etc.), but does not simulate the rendering on the client’s screen or the execution of client side scripts. Thus Java applets, Javascripts etc. are not executed (running scripts of many users would also put excessive load on the client’s CPU).

For example a scripted inclusion of a banner ad is not processed and thus the image request is not sent to the server. The implication is that the performance effect of such client side portions of the web application cannot be measured by this webservice loading/stressing technology.

Also any requests that are generated through Javascripts will also not be processed, use the Custom URL Script to add the these URLs manually.

Browser Simulation

If you use a proxy server, select **Use Proxy** and enter the address and **Port** of the proxy. If your proxy server requires authentication, select **Use Proxy User** and enter the **Username** and **Password**.

IMPORTANT: It is not recommended to run tests across proxy servers, because you will never know if you are actually testing the speed of your webservice or the speed of your proxy server.

A specific User Agent String can be sent to the server when **Use Agent** is selected. You may select a user agent string from the list or edit the string yourself.

To add your own parameters to the HTTP request headers, enable **Addtl. Headers** and enter your data in the text control.

You may set a maximum timeout for finishing the HTTP requests by selecting **Use Timeout**. Enter the timeout in seconds. A good value to start with is 60 seconds (since no human user would likely wait longer than that).

To throttle the data rate through which a user accesses the server, enter a value in kbit/s for **Simulate Maximum Data Rate for Client**. This can be used to simulate users accessing the webservice through modem lines (e.g. 50 kbit/s).

If your webservice or web application requires cookies, select **enable cookies**. The cookies are shown in the detailed log file when **Show Cookies in Log** is selected. The cookies are stored for each user and resent to the server for the following requests until a cookie invalidation is sent by the server.

Recursive Browsing

The features in this group should only be used on powerful client systems, especially for testing with high load conditions because the HTML of each request has to be parsed completely to identify image URLs, link URLs etc. Recursive Browsing features can result in considerable processor load on the client machine, which could result in inexact readings for the performance of the target server. Keep an eye on the processor load of the client during testing and run the test with and without these features. Compare the results to determine your client machine recursive browsing testing threshold.

By selecting **Download Image URLs** you instruct Webservice Stress Tool to parse all tags from the HTML code and send a request for each IMG URL to the server as soon as the complete HTML is received. If an image is used several times on the page, it is requested only once.

Enabling **Download EMBED, OBJECT and FLASH** also downloads these objects.

Select **Show Images in Log** and **Show Objects in Log** if you want to have a log file entry for each image in the detailed log, otherwise only one entry is generated stating how many images have been found and requested.

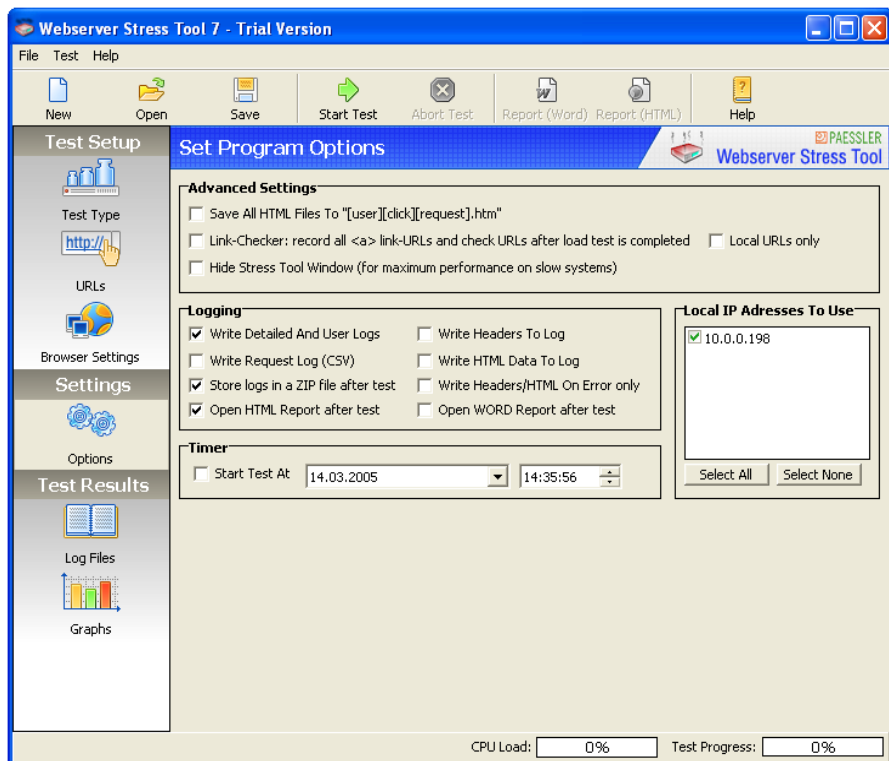
If your site uses HTML Frame tags you must select **Download Frames/IFrames**. Webrserver Stress Tool then parses the HTML code for <FRAME> and <IFRAME> tags. For each Frame URL a request is sent to the server. If this frame is a frameset again, additional requests are made until no more frames are found.

Select **Show Frames in Log** if you want to have a log file entry for each frame in the detailed log, otherwise only one entry is generated stating how many frame have been found and requested.

Some sites use the Refresh meta tag as means of redirection. To follow these redirections select **Follow “Refresh” Meta Tag**. The HTML is parsed for <meta name=“refresh” content=“time;url”> tags. As soon as a tag is found, a request is sent to the server (the time value is not used). Note: HTTP header redirects are always processed.

Setting Program Options

On this tab you can edit various program options:



Advanced Settings

The HTML results of all requests can be written to a disk file by selecting **Save all HTML files**. The filenames are built from the user number and the user’s click and request number. Note: Use with caution, this option can use a lot of CPU resources on the test client.

The **Link-Checker** stores all unique URLs from all requested HTML pages during the test run and tests all these URLs for broken links after the stress/load test is finished. The results can be found in the log files.

On slow client machines it might help to enable **Hide Stress Tool Window** to squeeze out a little more testing power (mainly because it makes sure no CPU cycles are used for screen updates.

Logging

Webserver Stress Tool always writes a summary logfile during test. For more detailed logfiles, enable **Write Detailed and User Logs**. A detailed log (for the entire test) and an individual log for each user's activity will be written to disk.

Please be aware that for high traffic load tests with hundreds or even thousands of users, detailed logging can have a serious impact on the performance of the testing client and thus the measured values can be incorrect. It is always a good idea to run heavy tests with and without detailed logging to compare the results, specially keep an eye on the CPU load of the client.

Using **Write Request Log (CSV)**, an additional machine readable logfile can be created that has one line of data per request of the test. This option is good if you need to process the results yourself. Note: The request log can also affect test client performance.

Choose **Store logs in a ZIP file after test** to store all the resulting logfiles as well as the configuration files of the test into one ZIP file for later reference. The file will be stored in the “zipped logs” subfolder of the EXE’s path and will show the date and time of the test in the filename.

You can immediately open the test report in your web browser or word if you enable **Open HTML Report after test** or **Open WORD Report after test**.

Webserver Stress Tool can also write all received data to the log file, select **Write Header to Log** for all data in the HTTP headers and **Write data to Log** for all HTML data of the requests.

When using **Write on Error Only**, only the data of requests that result in an error are written to the log.

Local IP Addresses to use

If the machine on which you run Webserver Stress Tool has more than one IP address, you may select which IP address should be used to simulate the requests. We have found that for most situations for HTTP load/stress tests you usually do not need to have more than one IP address, as the server answers all requests in the same manner regardless of the IP addresses. Only if your website or application uses the IP address to follow the sessions of a user etc. is it necessary to use more than one IP address.

If more than one IP address is selected, the IP addresses are used in a “round robin” manner for each simulated user. As the first user uses the first selected IP address for all his requests, the second user uses the second IP address, etc.

Timer

Using the **Start test at** feature you can postpone the start of the test to a specific date and time.

Performance Tips&Tricks

Finding the Bottleneck of Your Test Setup

When running load tests on a webserver you must make sure that you do not hit a performance limitation of your test client or your network.

The best way to find these limits is to run a ramp test with twice or three times the load you want to test with (or even more) and inspect the **Test Client's Health** graph afterwards.

The graph for **Network Traffic** and **local CPU Usage** should ramp up with the increasing number of users. When either one hits a plateau you have found your limit - or the limit of the server.

E.g. if you are using a 100 Mbit network you may see the Network Traffic graph hitting the 100 Mbit/s bandwidth limitation of your network hardware.

To distinguish between client/network and server issues it is a good idea to also monitor the CPU Load/Network of the server which will also help to find out what the bottleneck is. If Webserver Stress Tool already indicates a limit, but your server is more or less idle, you need a machine with more testing power.

Also keep any eye on the **Protocol Times** graph. Under heavy loads sometimes the **Time for local socket** can rise sharply (above 10-50 ms) which also indicates a performance bottleneck.

Network Issues

For load and stress tests, the network connection between the test client and the server is critical. For the connection between the server and the test client you must provide the full bandwidth that an equal number of real users would use when accessing your server!

This means that you obviously can not conduct a serious load test with 500 users requesting a 5 MB file over a single 56kb modem connection.

Additionally, if your are running the test from a remote location, the number and the performance of the hops (router/firewalls etc.) can influence the test. The optimum testing environment is to run the server and client within the same networked environment (i.e., on the same LAN).

For heavy load testing, it is the best to connect both the client and server to a high performance network switch. Since Webserver Stress Tool on a fast PC can easily

work with more bandwidth than a 100 Mbit LAN can deliver, even a Gigabit Ethernet may be a good idea.

For tests over internet connections like T1, DSL etc. you have to make sure that the amount of data created by your tests does not exceed the bandwidth of these connections. Use a bandwidth monitoring software like PRTG Traffic Grapher to monitor the bandwidth usage (www.paessler.com/prtg)

Usually for performance and smaller load tests a leased line with 500 kb/s or more should be enough, but more bandwidth will always give you more reliable results. Furthermore, you have to make sure that the “travel time” of the data is far below the request times you measure. Otherwise, measured values will be unreliable.

Everything below a 300 kb/s connection should be considered vague testing, although it can give good results under some circumstances, e.g. for long running web server scripts that only produce very little HTML code. The same applies for modem connections.

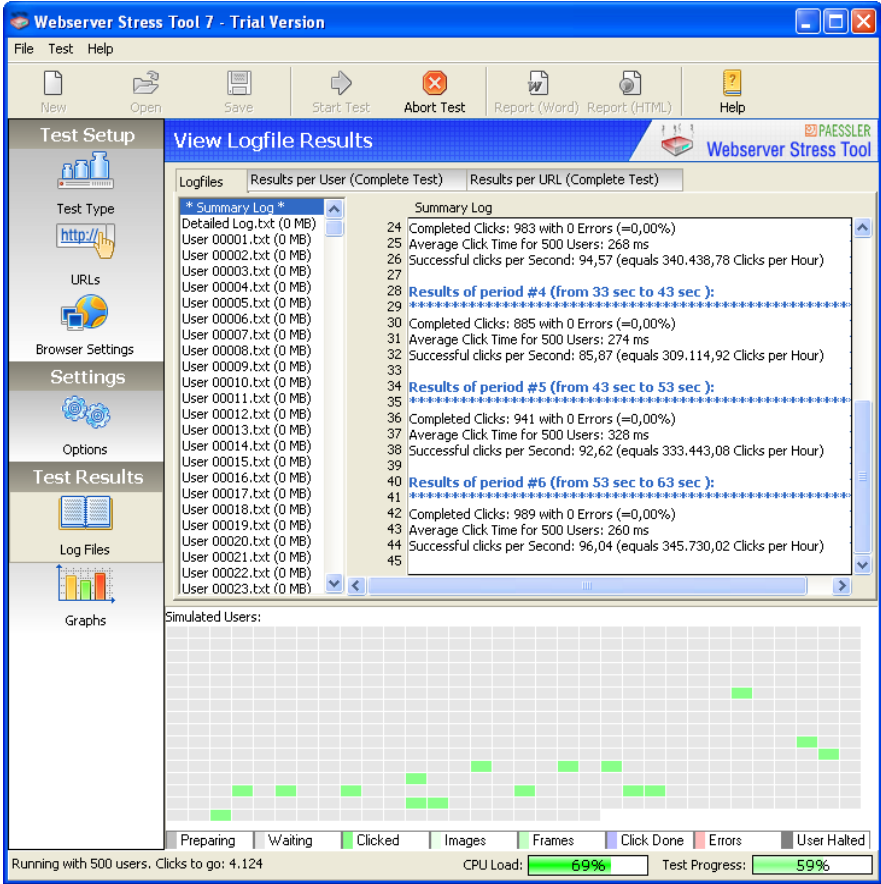
Test Client Issues

For high loads (>250.000 clicks/h) a client machine with multi processor (or at least Hyperthreading) is recommended.

It is also recommended to frequently defragment the disk drive that Webserver Stress Tool is using for the logs, because the high number of files growing steadily in small chunks can cause serious fragmentation.

Running the Test

After setting all desired settings for Load Pattern and Browser Settings, click on **Start Test** to make Webserver Stress Tool begin executing the test.



During the test you navigate through all settings pages, but you can not change the program and test settings.

You can however look at the test results already during the test.

As long as the test is active, there is a graphical view of the simulated users at the bottom of the window. Each user is shown by a rectangular area with a color showing the status of the user. This graph is updated every few seconds and will therefore not show all possible states for all users (that would slow processing

down). But nevertheless, this visualization provides a good illustration of what's going on in the test.

Also watch the status line at the bottom of the window for status information about the test.

By clicking **Abort Test** you can stop the test at any time.

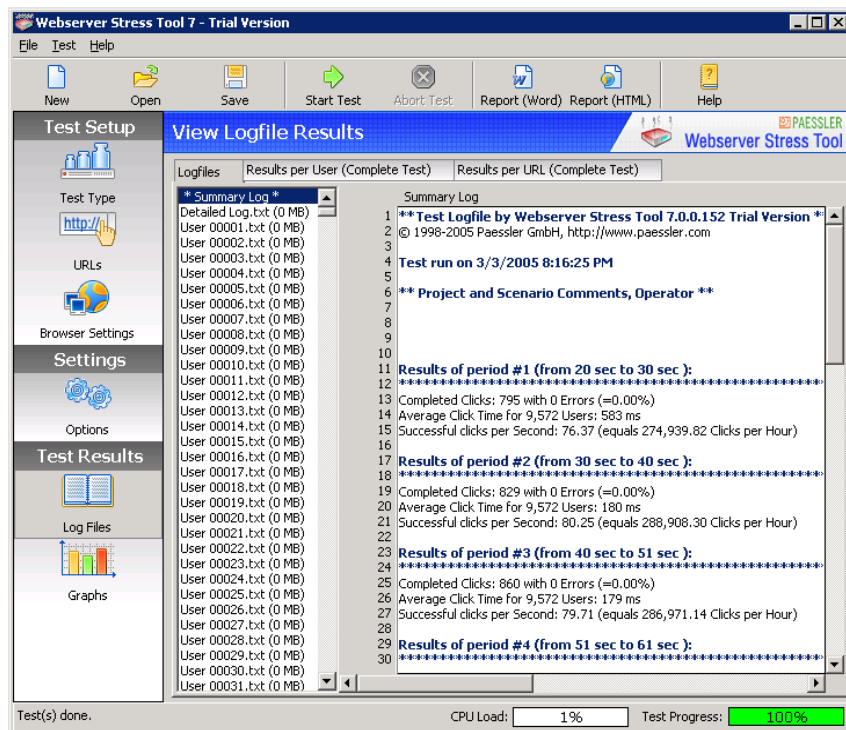
When the test is finished, the system will notify you with an audible sound (a beep). You can then review the results.

If you have enabled **Store logs in a ZIP file after test** in the options all results have been stored into one ZIP file for later reference.

As soon as the test is finished you will see the report in your web browser or in Word if you have enabled **Open HTML Report after test** or **Open WORD Report after test** in the options.

Reviewing Logfile Results

Click on **Log Files** to open the logfile browser.



On the left you will find a list of available logfiles. Simply click one of the entries to view the contents on the right.

If you enabled **Write HTML Files to disk** you can also select all the HTML files here.

All logfiles are saved to the "logs" subdirectory of the EXE's directory (usually C:\Program Files\Webserver Stress Tool 7\logs).

Summary Log and Detailed Log

There are two main log files: The **Summary Log** and the **Detailed Log**.

The summary log contains only the most important results:

- Time and Date of test
- Short results for all periods
- Short results of complete test
- Glossary

The detailed log (must be enabled on the options page) contains all of the Summary Log information and:

- Test Setup Data (URLs, number of users etc.)
- Test process information (e.g. waiting for timer)
- Detailed results for all periods
- Failed Requests
- Results of complete test
- Glossary
- Locations where the logfiles were saved to

This log file can grow very large. Depending on your operating system, Webserver Stress Tool may not be able to show the logfile. If this is the case, please use an external editor.

Note: large log files can not be opened on Windows 95/98/ME machines.

The detailed log file and the user's logfiles are written to the disk almost instantly during the test and thus can be helpful in diagnosing problems in the event of an abnormal program termination.

The summary log is written to the disk at the end of the test.

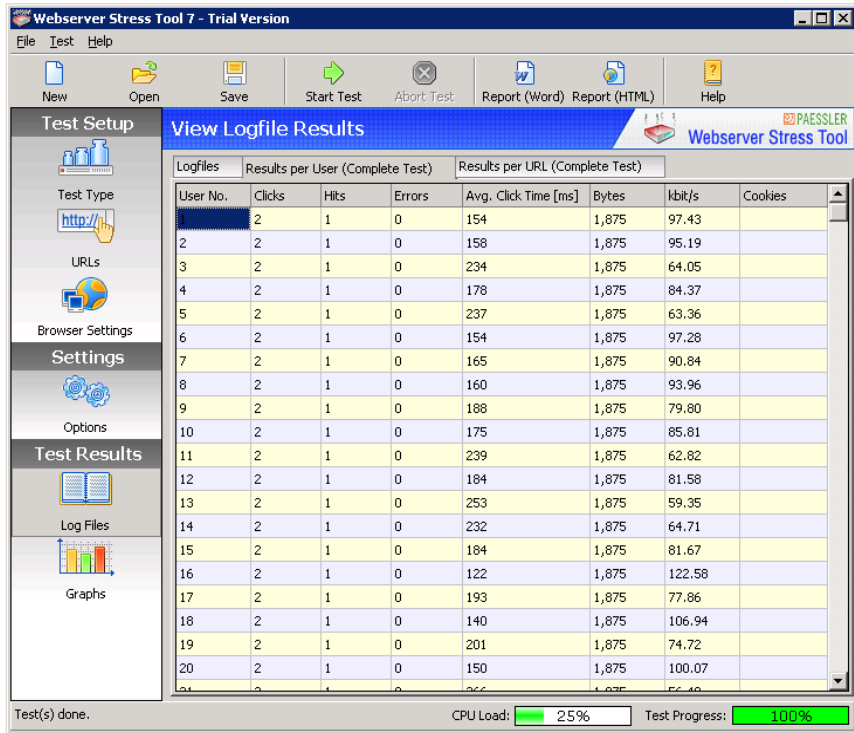
User Logs

If **Detailed Log** is enabled on the options page, Webserver Stress Tool writes a logfile for each user. This logfile contains:

- Activity log and data of all clicks, frames, images, redirects, requests etc.
- Time to first byte, Time to connect and similar data of each request
- Optionally all header data, HTML data, cookie data, image URLs and frame URLs
- Select a user log in the list of logfiles by clicking on it with the mouse and the file will be shown on the right portion of the window.

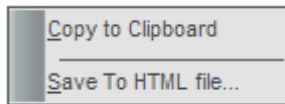
Results per User

The page **Results per User (Complete Test)** shows the resulting numbers for each simulated user:



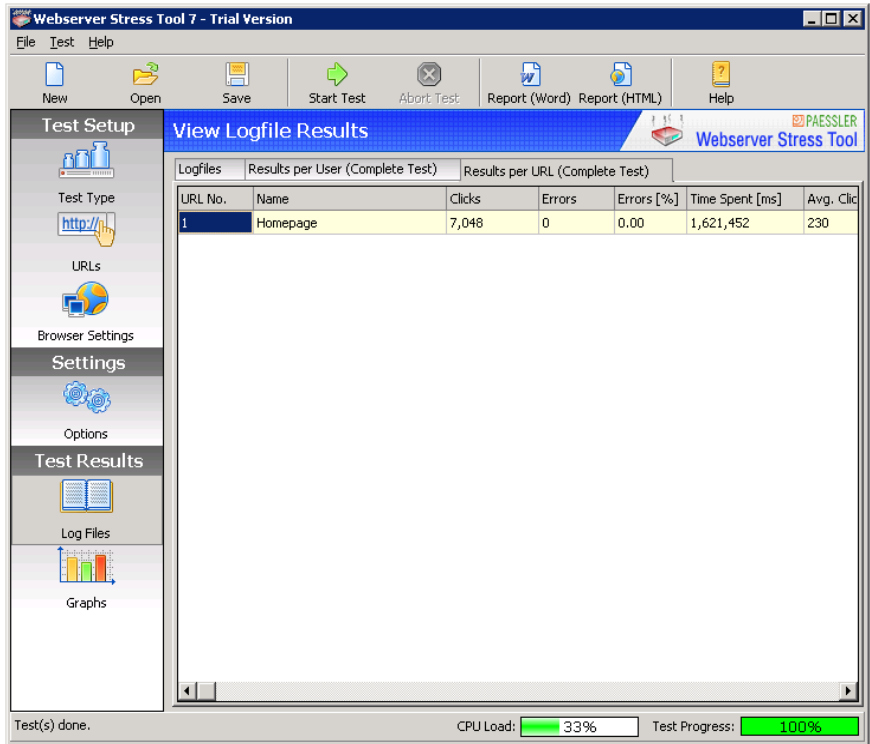
The data shown is the data aggregated over the complete test.

Right click the table for a context menu and you can copy the table to the clipboard or save it to a file:



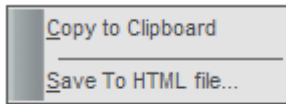
Results per URL

The number of hits, errors and time usage of each URL is shown on the page **Results per URL (Complete Test)**:



The data shown is the data aggregated over the complete test.

Right click the table for a context menu and you can copy the table to the clipboard or save it to a file:



Analyzing Graphical Results

This section describes the various graphs that are created during the test.

Graph Basics

Most graphs use the time since the start of the test as the horizontal axis.

Several graphs use more than one vertical axis, the secondary axis are shown on the right side of the chart.

For ramp tests the number of users that were active at a given moment in time is shown on the top of the graph. This axis is not linear because Webserver Stress Tool ramps to the highest number of users at 80% of the given test time,

Usage of the Graphs

Hiding Graph Lines

Using the checkboxes in the graph's legends you can hide/unhide individual lines from the chart.



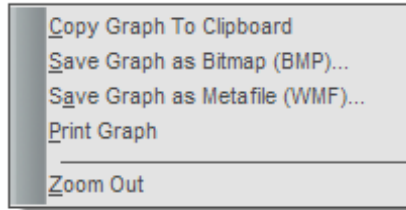
Zooming/Panning Graphs

You can zoom any graph by left click on the graph and dragging the mouse from top-left to bottom-right of the area you want to zoom into. Drag the mouse from bottom-right to top-left to zoom out again (or use the context menu to do so).

After you have zoomed into a graph you can right click on the graph and then move/pan the the chart.

Graph's Context Menu

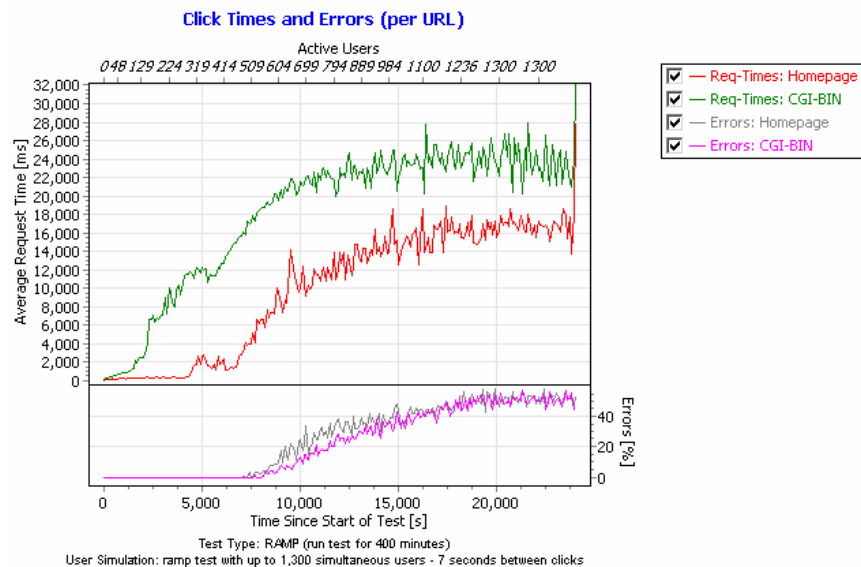
By using each graph's context menu (right mouse click on the graph) you can copy the graph to the clipboard, save it to disk as an image file or print it out.



Graph Click Times & Errors (per URL)

This can be considered the most important chart because it shows the average times and the rate of errors that the simulated users have experienced when downloading pages during the test.

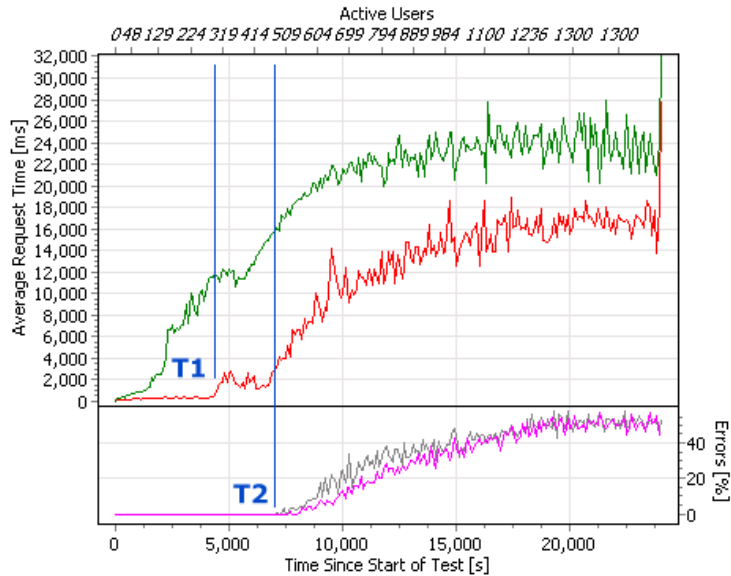
For each URL, this graph shows the request times of clicks and the percentage of errors (in the lower part of the chart). If you enable “download images” there are two more lines for each URL showing the average request times and errors for the images,



This sample graph shows the results of a 400 minute ramp test with up to 1,300 users accessing two URLs of a webserver every 7 seconds. One URL is a static HTML file (Homepage) and the other URL is a CGI script.

We can see that with the rising number of users the request times of the CGI script (green line) increase much faster than the request times for the static HTML page (red line).

In fact until about 300 simultaneous users (marked with “T1”) the request times for the static file don’t change much at all.



Then as the number of users crosses the 500 mark (marked with “T2”) we can see that the first requests produce errors. The graph of the percentage of errors (gray and pink line) goes up from 0% and keeps rising up to 50% until the end of the test.

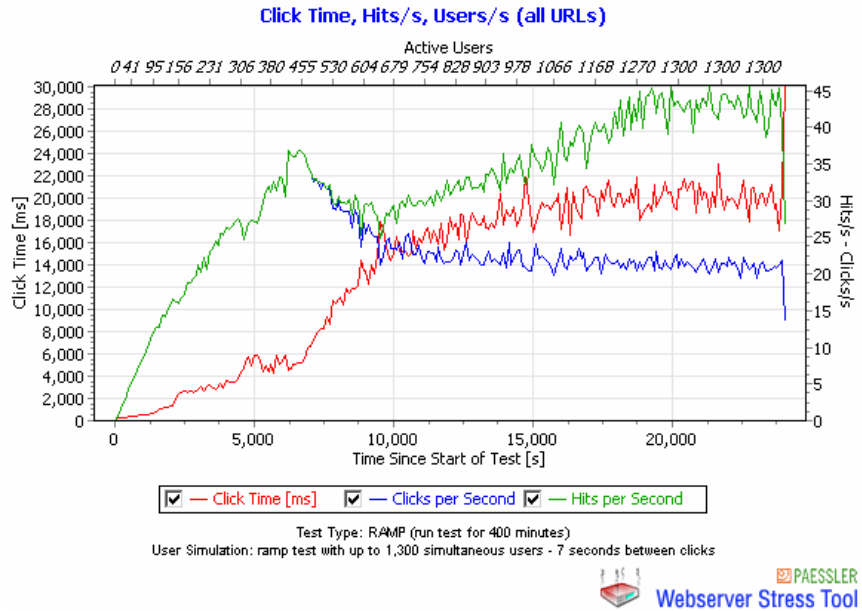
We can conclude that this server can support about 80-100 users clicking either link every 7 seconds with an average click time of 2 seconds. With more than 100 users the request times (especially those of the CGI) increase substantially.

This server can not support more than 500 users because with higher loads up to 50% of the requests produce errors.

Graph Click Times, Hits/s and Clicks/s

This graph shows the average time a user waited for his request to be processed (including redirects, images, frames etc., if enabled), the hits per second and the users per clicks. The difference to the graph above is that this time the values are calculated for all URLs together.

The following graph shows the results of the same test as in the previous section:

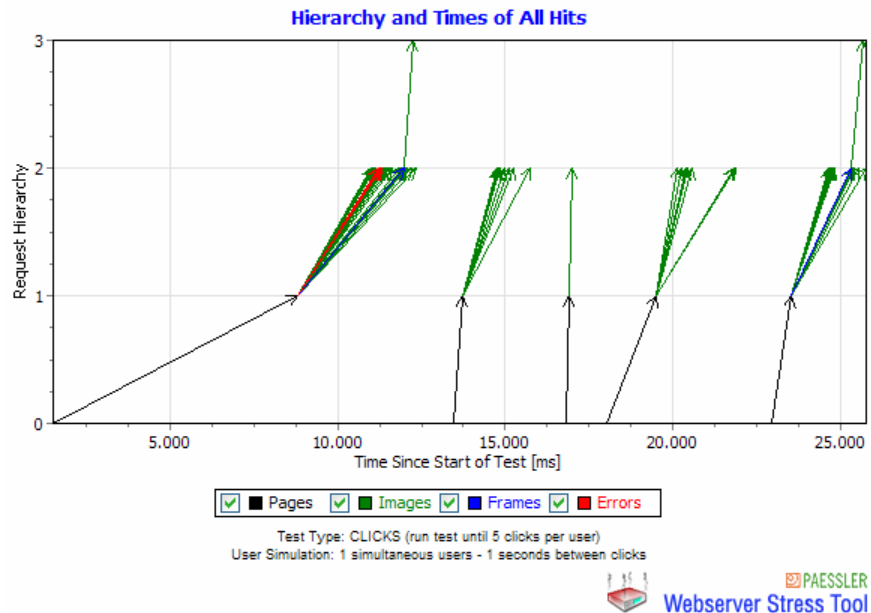


We can see that with more than 500 users the two lines for “clicks per second” (blue) and “hits per second” (green) differ more and more. The reason is that hits includes requests that produce errors, but clicks are only calculated from the requests that were successful.

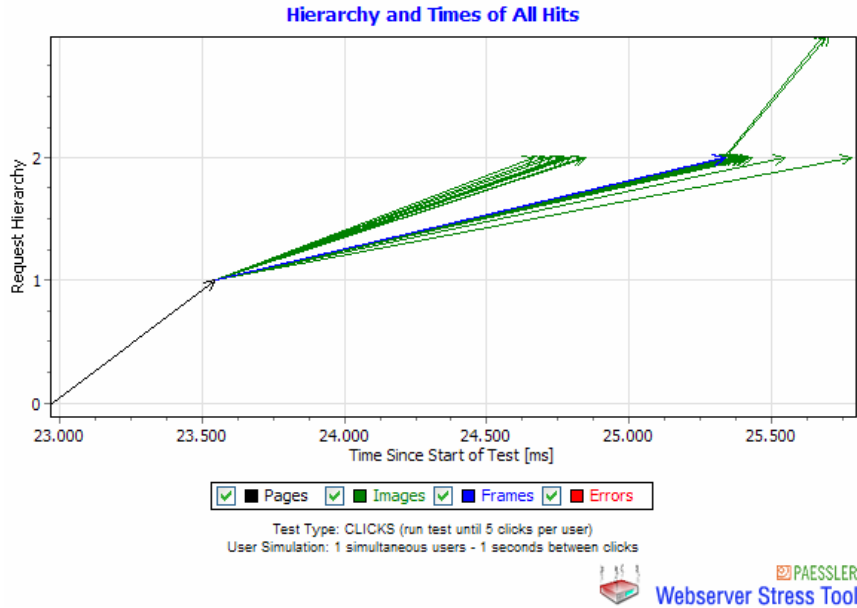
Graph Hierarchy

For each simulated request that Webservers Stress Tool sends to the server, one arrow is shown in this chart.

Each arrow represents one hit (i.e. one HTTP request). The black arrows are pages (i.e. HTML files), the green arrows represent images, the blue arrows show frames and the red arrows show failed requests.



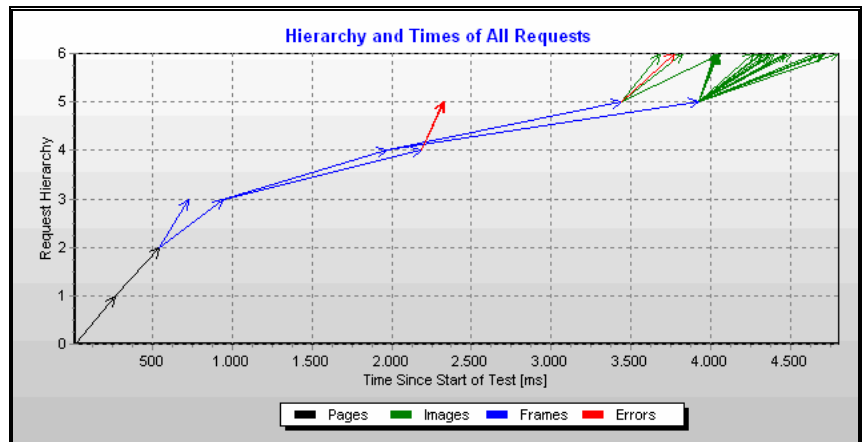
This sample chart shows clicks to several URLs with HTML pages (single black arrow) and pages with frames and images (black arrow with blue arrows). There are also some failed requests (red arrows).



The longer a request took, the further right the arrow ends. As soon as the HTML text of a page request is received, the images are requested from the server and shown in the chart with the green arrows.

Notice the red arrows which represent failed PAGE requests.

Here is an older example of a hierarchy graph:

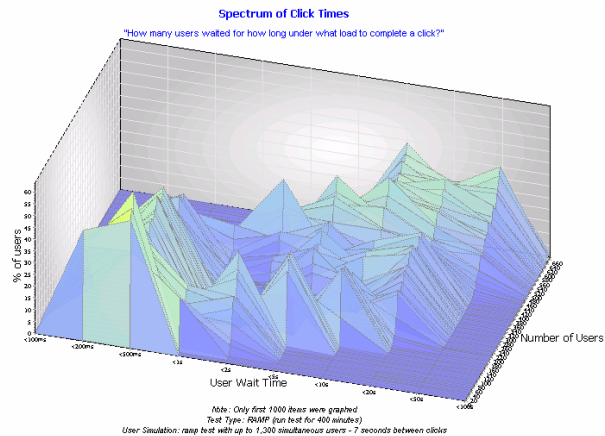


It shows the request hierarchy for one user to a website. The website has a homepage URL such as “www.company.com“ which redirects to a frameset. In this view, the arrow is the first request to the company URL. The request is then redirected to a frameset page (second arrow), which consists of several HTML pages/frames (blue arrows). The html pages of each frame then has their images (green and red arrows).

In total a visitor of this webpage needed at least 5 seconds for the complete page to load. That’s very slow...

Graph Spectrum of Click Times

This graph shows the distribution of user wait times for each run in the test.

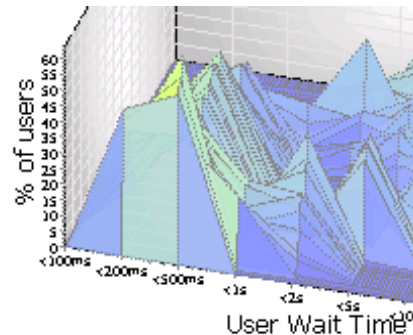


PAESSLER
Webserver Stress Tool

This sample graph shows the results of a Ramp Test. The three axis are:

- Vertical: percentage of users
- Horizontal: user wait time
- Depth: Number of users

At the beginning of the test (first bars at the front of the chart) most users get request times below 2 seconds.

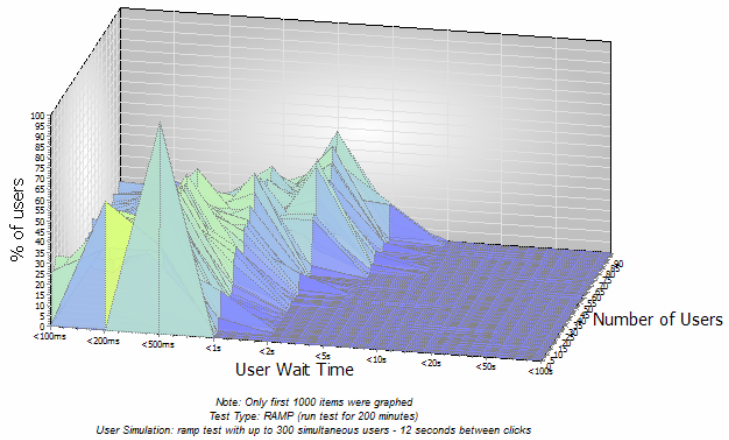


With more and more users accessing the server the request times deteriorate, the bar's maximum is moving from left to right with increasing depth.

In this other sample the effect is still visible, but the request times at the end of the test are still below 5s.

Spectrum of Click Times

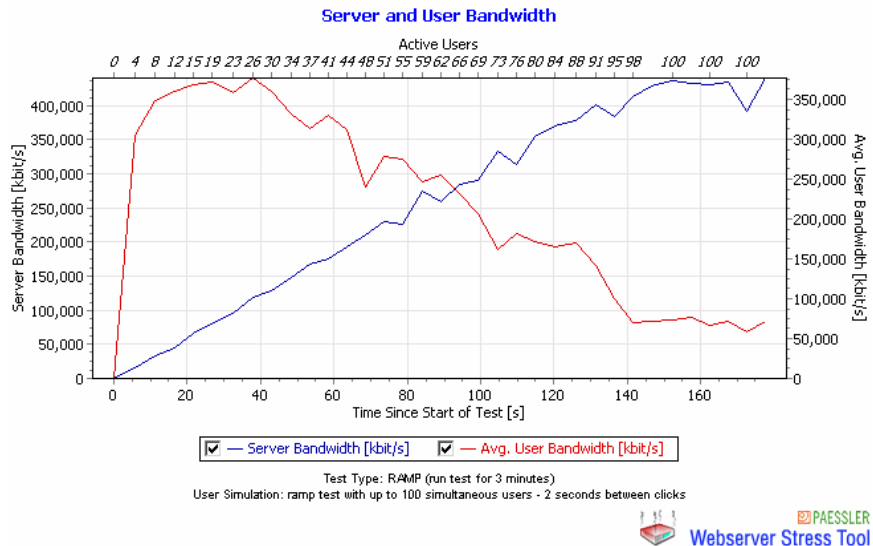
"How many users waited for how long under what load to complete a click?"



The affect of this test on capacity planning is clear. Consider that the maximum response time goal for each user should be ten (10) seconds or less. With this goal in mind, you have to make sure that your graph has its maximum at the "<10s" reading or better – for the number of users you want to be able to support.

Graph Server and User Bandwidth

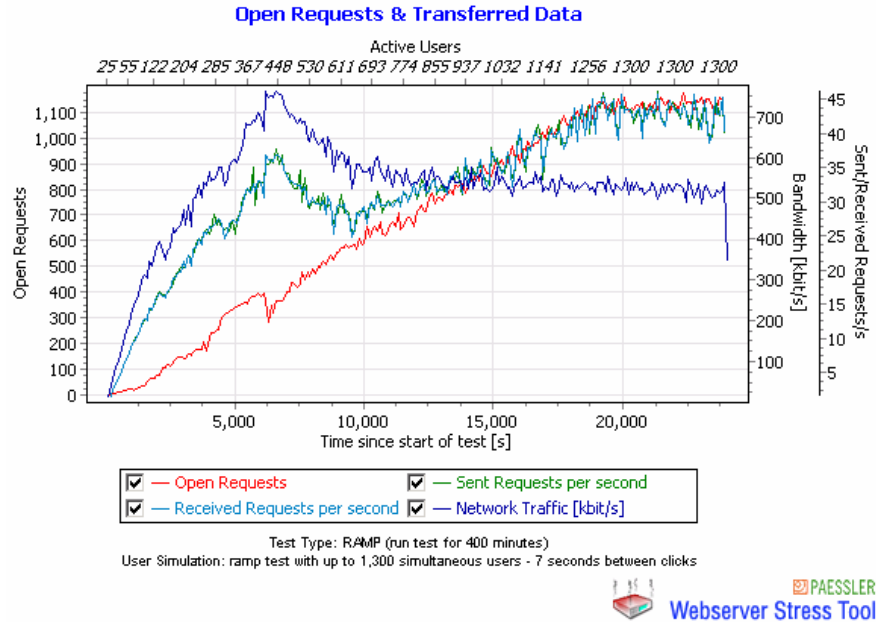
This graph displays the bandwidth the server was able to deliver (as a total) as well as the average bandwidth that was experienced by the simulated users:



In this graph we can see that the average bandwidth available per user goes down from 360 Mbit to 80 Mbit when the number of users climbs from 1 to 100 users.

Graph Open Requests and Traffic

This graph shows the number of open requests as well as the number of sent and received requests in comparison with the network traffic:

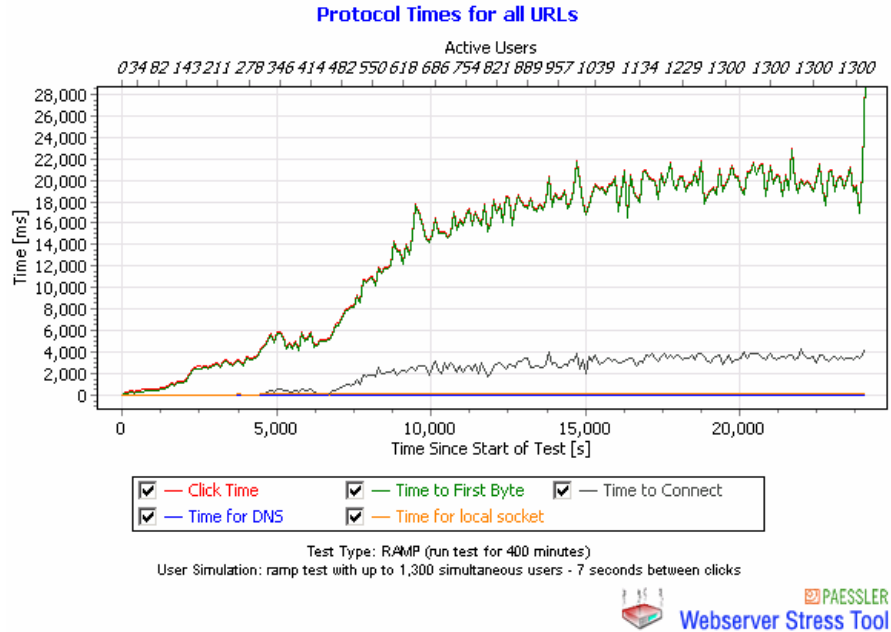


Graph Protocol Times

An HTTP request consists of several stages. First, the webserver name has to be resolved into an IP address using DNS (Time for DNS), then an IP port is opened on the server by the client to send the request header (Time to Connect). The server then answers the request (Time to First Byte) and sends all data. When all data is transferred, the request is finished (Click Time).

Also in this graph a line is shown for the “time for local socket” which is the time that Webserver Stress Tool needed to acquire an open socket from the IP stack of the machine it runs on. For usual test this value should always be in the lower millisecond area (1-30 ms). For extreme traffic tests this value can rise above 50-100 ms which is a sign that the performance limits of the local machine have been reached.

The average value of these five readings are displayed in this graph:

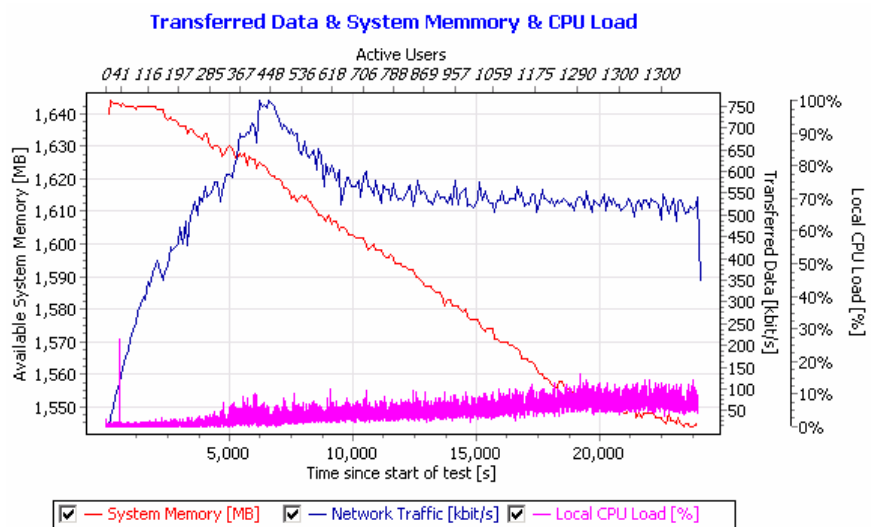


Graph Test Client's Health

For this last graph Webserver Stress Tool constantly measures vital parameters of the machine it runs on. It can be helpful to find out if the limits of the test client have been reached.

Especially the line for the CPU Load (pink) should be well below 100%. If you constantly hit values above 90% for the CPU load the test results may be incorrect.

Also the network traffic (blue line) should be below the physical limits of your connection to the server.



Creating Reports

Websaver Stress Tool offers two methods to export results.

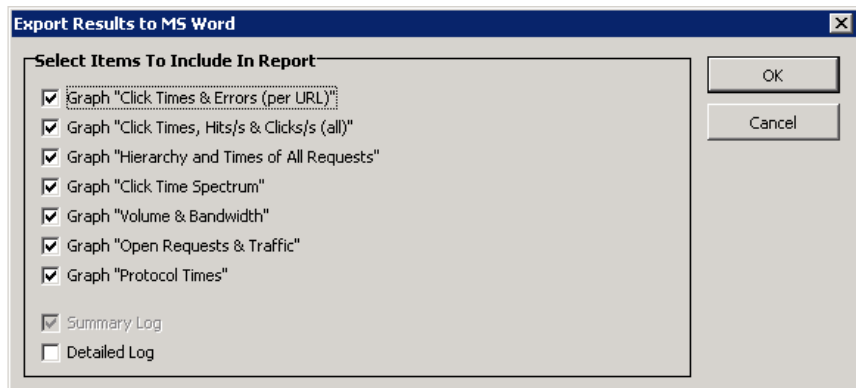
You can export all resulting information into a MS Word document (MS Office must be installed) and you can create a number of HTML files.

These reports can be created manually or automatically as soon as the test is finished. Please enable **Open HTML Report after test** or **Open WORD Report after test** in the options.

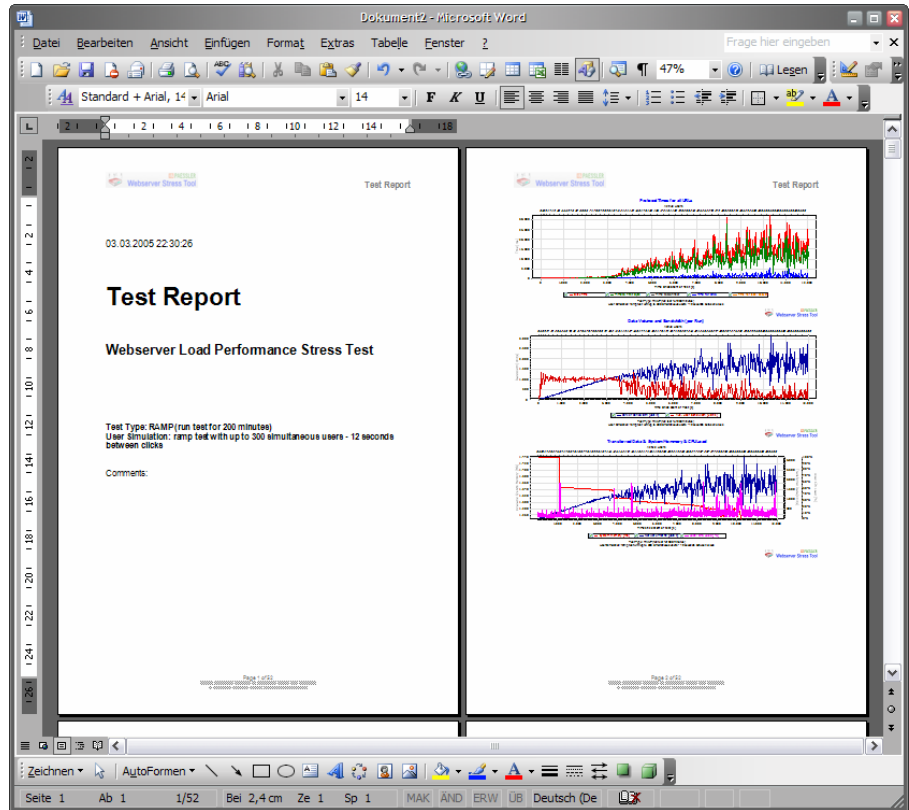
Report (Word)

The best way to store all results of a test into one file is to create a DOC file.

If you have Microsoft Office installed on the client machine, click on **Report (Word)** after a test is finished:



Select what data you want included in the report. As soon as you click **OK**, Microsoft Word is started using OLE and the report is built. A few seconds later you can edit, print and save the report using all the normal functions of Microsoft Word.



Report (HTML)

Click on **Report (HTML)** to create a set of HTML and images files with the results of the test. Choose an item from the menu in the left frame to navigate through the results.

Note: The files of the HTML report are deleted whenever Webserver Stress Tool is (re-)started or when a new test is started.

Webserver Stress Tool - Report - Microsoft Internet Explorer

Adresse <D:\Webserver Stress Tool 7\logs\report.html>

Webserver Stress Tool

Test Report

Settings

- [Test Configuration](#)
- [URLs](#)
- Results and Graphs**
- [Click Times and Errors](#)
- [Click Time, Hits/s & Clicks/s](#)
- [Protocol Times](#)
- [Hierarchy](#)
- [Open Requests & Traffic](#)
- [System Health](#)
- [Volume & Bandwidth](#)
- [Click Time Spectrum](#)
- [Results per User](#)
- [Results per URL](#)

Logfiles

- [Detailed Log.txt](#) (53 MB)
- [Summary Log.txt](#) (0 MB)
- [User 00001.txt](#) (1 MB)
- [User 00002.txt](#) (1 MB)
- [User 00003.txt](#) (1 MB)
- [User 00004.txt](#) (1 MB)
- [User 00005.txt](#) (1 MB)
- [User 00006.txt](#) (1 MB)
- [User 00007.txt](#) (1 MB)

Project and Scenario Comments, Operator

Test Setup

Test Type: RAMP (run test for 400 minutes)
 User Simulation: ramp test with up to 1,300 simultaneous users - 7 seconds between clicks
 Logging Period: Log every 100 seconds

URLs

URL Sequencing: Users always click the same URL (to spread load evenly on all URLs, set number of users to a multiple of the number of URLs!)
 URLs: [Click here](#)

Browser Settings

Browser Simulation: User Agent: Mozilla/5.0 (compatible; Webserver Stress Tool 7; Windows)
 Browser Simulation: HTTP Request Timeout: 120 s

Options

Logging: Write detailed log(s)
 Timer: not enabled
 Using Local IPs: 10.0.0.138

Client System

System: Windows 2000 V5.2 (Build 3790) , CPU Proc. Type 586 (Rev. 772) at 2800 MHz,
 Memory: 1738 MB available RAM of 2146 MB total physical RAM, 3869 MB available pagefile, 4078 MB free disk space on C:
 Client's MAC address: 00-50-56-C0-00-08
 Client's MAC address: 00-50-56-C0-00-01
 Client's MAC address: 00-C0-9F-3F-28-26
 Time measurement resolution: 0.0035710000 usec, clock runs at 2,800 MHz

Test Software

Webserver Stress Tool: 7.0.0.152 Trial Version

Fertig My Computer

Additional Features

Working with Different Test Scenarios

Each test scenario can be saved to disk using the menu option **File|Save Scenario as**. Each scenario consists of two files— a .INI file and a .DAT file.

Using **File|Open Scenario File** the scenarios can be reloaded again later.

Command Line Interface

Websserver Stress Tool offers a command line interface to automate testing.

By using “webstress7.exe scenarionfile.ini” as command line input, you instruct Websserver Stress Tool to load the scenario, run the test and export the results into MS Word. The report and the log files are saved with a filename that includes the name of the scenario file.

Running Several Tests at Once

To run several tests at once you must first make sure that only one instance of webstress7.exe is running in one folder. That means that if you want to run several copies of Websserver Stress Tool at once you must make sure that each EXE runs in its own folder. Simply copy the \program files\websserver stress tool 7 folder as often as you need it.

Then create a CMD batch file like the following:

```
Start \folder1\webstress7.exe \folder1\1stszenario.ini
Start \folder2\webstress7.exe \folder2\2ndszenario.ini
Start \folder3\webstress7.exe \folder3\3rdszenario.ini
```

Using Tokens

Some servers generate a session ID when a user logs into a site that has to be placed into all subsequent URLs. You do this with Websserver Stress Tool by placing the token into the HTML code of the page following the login (or anywhere else) using the following code:

```
<!--TOKENyoursessionidhereTOKEN-->
```

Any code inside a HTML page between “<!--TOKEN” and “TOKEN-->” is stored for each user individually and can be resent in the next URL(s) or POSTDATA by using the placeholder @ @ @.

Tips and Tricks

Check out the Paessler Knowledge Base

It is always a good idea to check out the Paessler Knowledge Base which includes a number of technical articles about Webservers Stress Tool:

<http://www.paessler.com/support>

Recording HTTP URLs for Complex Web Applications

For load and stress testing tools like Webservers Stress Tool you need a list of URLs to be tested. For complex applications using several frames or popups it can be quite complicated to come up with this list of URLs. The solution is using an HTTP logging proxy.

To load, performance, and stress test an HTTP application or web server using a HTTP request generator like Webservers Stress Tool you must have a list of URLs the testing tool generates requests for. Since Webservers Stress Tool does not simulate all the client processes (e.g. javascripts) it cannot come up with this list of URLs.

Why doesn't Webservers Stress Tool simulate everything that happens inside the browser: The answer is PURE POWER: Simulating the javascripts of hundreds or even thousands of users would make the test client machine so slow, you wouldn't get a stress or load test for the server anymore!

The built-in URL recorder of Webservers Stress Tool works fine with simple web front ends. But as soon as more than one pop up, several frames, AJAX features or Flash are used the recorder hits its limits. The solution is to use an **HTTP logging proxy**. You configure Internet Explorer (or any other browser) to access the web application to be tested through this proxy and the proxy logs all HTTP requests (URLs, POSTDATA etc.) so you can later extract the URLs and feed them into Webservers Stress Tool.

Good tools to start with are Fiddler Debugging Proxy from Microsoft (<http://www.fiddlertool.com/fiddler/>) or HTTPTracer from Lazy Dog Software (<http://www.lazydogutilities.com/traceprev.htm>).

After you have your set of URLs it is very likely that you have to use placeholders to replace the "per user" data in these URLs (sessions ids, usernames, passwords etc.). Use the placeholders and data merging of Webservers Stress Tool to accomplish this.

Appendix

Script Syntax for URL Scripts

Basic syntax

- sub .. end and function .. end declarations
- byref and dim directives
- if .. then .. else .. end constructor
- for .. to .. step .. next constructor
- do .. while .. loop and do .. loop .. while constructors
- do .. until .. loop and do .. loop .. until constructors
- ^ , * , / , and , + , - , or , <> , >= , <= , = , > , < , div , mod , xor , shl , shr operators
- try .. except and try .. finally blocks
- select case .. end select constructor
- array constructors (x:=[1, 2, 3];
- exit statement
- access to object properties and methods (ObjectName.SubObject.Property)

Script structure

The script must contain the 4 necessary scripts defined as subs (see Using Custom URL Scripts Section).

Identifiers

Identifier names in script (variable names, function and procedure names, etc.) follow the most common rules in basic: should begin with a character (a..z or A..Z), or '_' , and can be followed by alphanumeric chars or '_' char. Cannot contain any other character or spaces.

Valid identifiers:

```
VarName
_Some
V1A2
_____Some_____
```

Invalid identifiers:

```
2Var
My Name
Some-more
This,is,not,valid
```

Assign statements

Assign statements (assign a value or expression result to a variable or object property) are built using "=".

Examples:

```
MyVar = 2
Button.Caption = "This " + "is ok."
```

Character strings

strings (sequence of characters) are declared in basic using double quote (") character.

Some examples:

```
A = "This is a text"
Str = "Text "+"concat"
```

Comments

Comments can be inserted inside script. You can use ' chars or **REM**. Comments finish at the end of line.

Examples:

```
' This is a comment before ShowMessage
ShowMessage("Ok")
REM This is another comment
ShowMessage("More ok!")
'
And this is a comment
' with two lines
ShowMessage("End of okays")
```

Variables

There is no need to declare variable types in scripts, but you can optionally declare variables by using **DIM** directive and its name.

Indexes

Strings, arrays and array properties can be indexed using "[" and "]" chars. For example, if Str is a string variable, the expression Str[3] returns the third character in the string denoted by Str, while Str[I + 1] returns the character immediately after the one indexed by I.

More examples:

```
MyChar = MyStr[2]
MyStr[1] = "A"
MyArray[1,2] = 1530
Lines.Strings[2] = "Some text"
```

Arrays

Scripts support array constructors and support to variant arrays. To construct an array, use "[" and "]" chars. You can construct multi-index array nesting array constructors. You can then access arrays using indexes. If array is multi-index, separate indexes using ",". Arrays in scripts use a zero-based index.

Some examples:

```
NewArray = [ 2,4,6,8 ]
Num = NewArray[1] //Num receives "4"
MultiArray = [{"green","red","blue"} , [{"apple","orange","lemon"}]
Str = MultiArray[0,2] //Str receives 'blue'
MultiArray[1,1] = "new orange"
```

if statements

There are two forms of if statement: **if...then..end if** and the **if...then...else..end if**. Like normal basic, if the if expression is true, the statements are executed. If there is else part and expression is false, statements after else are executed.

Examples:

```
IF J <> 0 THEN Result = I/J END IF
IF J = 0 THEN Exit ELSE Result := I/J END IF
IF J <> 0 THEN
  Result = I/J
  Count = Count + 1
ELSE
  Done = True
END IF
```

while statements

A **while** statement is used to repeat statements, while a control condition (expression) is evaluated as true. The control condition is evaluated before the statements. Hence, if the control condition is false at first iteration, the statement sequence is never executed. The while statement executes its constituent statement repeatedly, testing expression before each iteration. As long as expression returns True, execution continues.

Examples:

```
WHILE (Data[I] <> X) I = I + 1 END WHILE

WHILE (I > 0)
  IF Odd(I) THEN Z = Z * X END IF
  X = Sqr(X)
END WHILE

WHILE (not done)
  (**Some code here**)
END WHILE
```

loop statements

Scripts support **loop** statements. The possible syntax is:

```
DO WHILE expr statements LOOP
DO UNTIL expr statements LOOP
DO statements LOOP WHILE expr
DO statement LOOP UNTIL expr
```

Statements will be executed WHILE expr is true, or UNTIL expr is true. if expr is before statements, then the control condition will be tested before iteration. Otherwise, control condition will be tested after iteration.

Examples:

```
DO
  K = I mod J
  I = J
  J = K
LOOP UNTIL J = 0

DO UNTIL I >= 0
  (**Some code here**)
LOOP

DO
  K = I mod J
  I = J
  J = K
LOOP WHILE J <> 0

DO WHILE I < 0
  (**Some code here**)
LOOP
```

for statements

Scripts support **for** statements with the following syntax: FOR counter = initialValue TO finalValue STEP stepValue statements NEXT. For statement set counter to initialValue, repeats execution of statement until "next" and increment value of counter by stepValue, until counter reaches finalValue. Step part is optional, and if omitted stepValue is considered 1.

Examples:

```
SCRIPT 1:
FOR c = 1 TO 10 STEP 2
  a = a + c
NEXT

SCRIPT 2:
FOR I = a TO b
  j = i ^ 2
  sum = sum + j
NEXT
```

select case statements

Script supports **select case** statements with following syntax:

```
SELECT CASE selectorExpression
  CASE caseexpr1
    statement1
  CASE caseexprn
    statementn
  CASE ELSE
    elsestatement
END SELECT
```

if selectorExpression matches the result of one of caseexprn expressions, the respective statements will be execute. Otherwise, elsestatement will be executed. Else part of case statement is optional.

Example:

```
SELECT CASE uppercase(Fruit)
  CASE "lime" (**Some code here**)
  CASE "orange" (**Some code here**)
  CASE "apple" (**Some code here**)
  CASE ELSE
    (**Some code here**)
END SELECT
```

function and sub declaration

Declaration of **functions** and **subs** are similar to basic. In functions to return function values, use implicity declared variable which has the same name of the function. Parameters by reference can also be used, using BYREF directive.

Some examples:

```
SUB HelloWorld
  (**Some code here**)
END SUB

SUB UppcaseMessage(Msg)
  (**Some code here**)
END SUB

FUNCTION TodayAsString
  TodayAsString = DateToStr(Date)
END FUNCTION

FUNCTION Max(A,B)
  IF A>B THEN
    MAX = A
  ELSE
    MAX = B
  END IF
END FUNCTION

SUB SwapValues(BYREF A, B)
  DIM TEMP
  TEMP = A
  A = B
  B = TEMP
END SUB
```

Additional Functions

Saving a String to a file:

```
a=savestringtofile("d:\temp\myfile",mystring)
if a<>0 then
  data.log="Could not write file (result="+inttostr(a)+")"
end if
```

Loading a file into a string:

```
data.postdata=loadstringfromfile("D:\temp\mypostdata")
```

Useful RFCs

The following Request for Comment (RFC) documents provide valuable general background information for web stress professionals.

W3C,

HTTP Specifications and Drafts,

<http://www.w3.org/Protocols/Specs.html>

Berners-Lee, T., Masinter, L. and M. McCahill,
Uniform Resource Locators (URL) ,
RFC 1738, December 1994.
<http://www.ietf.org/rfc/rfc1738.txt>

Berners-Lee, T., Fielding, R. and H. Frystyk,
Hypertext Transfer Protocol -- HTTP/1.0 ,
RFC 1945, May 1996.
<http://www.ietf.org/rfc/rfc1945.txt>

Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and T. Berners-Lee,
Hypertext Transfer Protocol -- HTTP/1.1 ,
RFC 2068, January 1997.
<http://www.ietf.org/rfc/rfc2068.txt>

Irvine, Gettys, Mogul et al.,
Hypertext Transfer Protocol -- HTTP/1.1 .
RFC 2616 (update to RFC 2068), June 1999,
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

Berners-Lee, T., Fielding, R. and L. Masinter,
Uniform Resource Identifiers (URI): Generic Syntax and Semantics ,
RFC 2396, August 1998.
<http://www.ietf.org/rfc/rfc2396.txt>

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A.,
Sink, E. and L. Stewart,
HTTP Authentication: Basic and Digest Access Authentication ,
RFC 2617, June 1999.
<http://www.ietf.org/rfc/rfc2617.txt>

S. Spero,
Analysis of HTTP Performance Problems,
<http://sunsite.unc.edu/mdma-release/http-prob.html>

Useful Books

The following books will be helpful during your efforts to test the performance and usability of your web applications and website.

Andrew B. King: Speed Up Your Site

Pearson Education, 2003

There is a time bomb on the web: user patience. It starts ticking each time someone opens one of your pages. You only have a few seconds to get compelling content onto the screen. Fail, and you can kiss your customers and profits goodbye. You can't count on fast connections either. Most of your customers are still sucking content through a 56K straw. You have to serve up greased lightning or they'll bail. That's why you picked up this book.

In it you'll learn how to cut file sizes in half. You'll trim (X)HTML, CSS, graphics, JavaScript, multimedia, and bandwidth costs. Real-world examples illustrate techniques with before and after code and percentage savings. After reading this book, you'll know how to make your pages literally "pop" onto the screen.

Order from Amazon.com:

<http://www.amazon.com/exec/obidos/ASIN/0735713243/paessler-20>

Splaine, Jaskiel: The Web Testing Handbook

STQE Publishing, 2001

The Web Testing Handbook is the definitive resource for testing Web sites and Internet-based applications. Many developers and testers are making the transition from traditional Client/Server, PC, and/or Mainframe systems to testing rapidly changing Web sites and applications.

The Web Testing Handbook can help make this transition easier by explaining these new technologies and suggesting test cases and techniques that can be included in a Web site's Functional, Performance, Compatibility, and Usability test plans.

Order from Amazon.com:

<http://www.amazon.com/exec/obidos/ASIN/0970436300/paessler-20>

Nguyen: Testing Applications on the Web

John Wiley & Sons, 2000

Written by a true authority in the field, Hung Q. Nguyen's Testing Applications on the Web is a nicely comprehensive guide to virtually every conceivable aspect of software testing. It's filled with must-have background information for any test engineer or manager who's testing thin-client systems.

Order from Amazon.com:

<http://www.amazon.com/exec/obidos/ASIN/047139470X/paessler-20>

Nielsen: Designing Web Usability

New Riders Publishing, 1999

Simply the bible for website usability.

Order from Amazon.com:

<http://www.amazon.com/exec/obidos/ASIN/156205810X/paessler-20>

Software License and Contact Information

Ordering Webserver Stress Tool

We offer various order and payment options. Please visit our website at www.paessler.com/order to learn more!

Copyright

Webserver Stress Tool is copyrighted © 1998-2007 by Paessler AG

Paessler AG
Hornschurchpromenade 7
D-90762 Fürth
Germany

Homepage: <http://www.paessler.com>

Support

Please visit <http://www.paessler.com/support> for available support options.

Consulting, Custom Versions of the Software

Paessler offers webserver stress test consulting and the development of custom versions. Please inquire via email to sales@paessler.com.

License/Usage Terms

PLEASE READ THESE GENERAL TERMS AND CONDITIONS CAREFULLY.

§ 1 TERRITORY

These General Terms and Conditions govern the use and maintenance of the Paessler Software for customers who use the software outside the Federal Republic of Germany. The use of the Paessler Software in the territory of the Federal Republic of Germany is governed by the "Allgemeinen Lizenzbedingungen der Paessler AG".

§ 2 DEFINITIONS

Site: Defined and/or restricted area (e.g. campus, premises) of which the diameter does not exceed 6,21 miles (10 kilometers), and which is used exclusively by the customer;

Customer: Contract partner who licensed the Paessler Software;

User: Person working with the software.

§ 3 PRE-CONDITION

PAESSLER AG IS WILLING TO LICENSE THE PAESSLER SOFTWARE TO THE CUSTOMER ONLY ON THE CONDITION THAT THE CUSTOMER ACCEPTS ALL OF THE TERMS CONTAINED IN THIS AGREEMENT.

§ 4 ASSIGNMENT

BY DOWNLOADING OR INSTALLING THIS SOFTWARE, THE CUSTOMER ACCEPTS THE TERMS OF THESE GENERAL TERMS AND CONDITIONS AND INDICATES THE ACCEPTANCE THEREOF BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THESE GENERAL TERMS AND CONDITIONS. IF THE CUSTOMER IS NOT WILLING TO BE BOUND BY ALL THE TERMS, THE CUSTOMER SELECTS THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL BE INTERRUPTED.

§ 5 LICENCES

Paessler grants to the customer a non-exclusive license to use the Paessler Software in object code form as described in these General Terms and Conditions.

1. "Commercial-Edition"

The "Commercial-Edition-Software" is designed for the exclusive use by the customer. For the grant of license a fee has to be paid.

The „Commercial-Edition“-License is non-transferable. Any attempt to share or transfer a licence without the consent of Paessler shall be a violation of this license agreement and international copyright laws, and will result in the forfeit of all benefits and rights as a user.

A. Single- and Multi-user license

The customer purchases a certain number of licences according to the confirmation of order. The customer is only allowed to install and use the maximum number of purchased licences simultaneously. This means the customer may only use the modules contained within the software for which the customer has paid a license fee and for which the customer has received a product authorization key from Paessler.

If the number of licenses allowed to be use simultaneously is exceeded, Paessler has to be informed and the exceeded number of installations have to be prevented through suitable means.

If the exceeded number of installations can not be prevented by organizational oder technical means, the customer is obliged to purchase the relevant number of licences.

B. Site- License

Paessler grants the customer the right and license to install and use the software on multiple computers for one or more users.

This license, however, is restricted to the use within one location (site). In the case of the software being used in other locations than those agreed upon, the customer is obliged to pay the license fee due for the location-license as a compensation.

C.Verification

The customer grants Paessler or its independent accountants the right to examine the customer's books, records, and accounts during the customer's normal business hours to verify compliance with the above provisions.

2. "Trial-Edition"

"Trial Edition" means a free-of-charge-version of the software to be used only to review, demonstrate, and evaluate the software. The Trial-Edition may have limited features, and /or will cease operating after a pre-determined amount of time, due to an internal mechanism within the Trial-Edition.

No maintenance is available for the Trial-Edition.

The software may be installed on multiple computers for private and commercial use.

The software may be transferred to third parties (e.g. on homepages or ftp-servers) as long as the program remains unchanged and is offered free of charge. A chargeable transfer of the "Trial-Edition" (chargeable download, CD in Magazines) requires previous written permission by Paessler.

§ 6 RESTRICTIONS

Software under these General Terms and Conditions is confidential and copyrighted. The title to the software and all associated intellectual property rights are retained by Paessler.

The customer may not modify, decompile, or reverse engineer said software.

The customer may make archival copies of the software.

§ 7 MAINTENANCE

Object of the Maintenance Agreement is the maintenance of the "Commercial-Edition-Software" according to the confirmation of order.

1. Scope of Maintenance

The customer is granted the opportunity of downloading the current version of the purchased Commercial-Edition-Software and requesting a license-key anew for the duration of the Maintenance Agreement.

The customer will receive support for the duration of the Maintenance Agreement.

2. Duration

The Maintenance Agreement has a duration of 12 months and may be prolonged.

3. Beginning

The Maintenance Agreement starts with the sending of the affirmation of contract. The affirmation includes the download-link and the licence-key of the software.

4. Prolongation

Prolongation may only take place until the end of the duration of the Maintenance Agreement.

§ 8 PRICE

All levies, dues, taxes, duties, and other charges shall be borne by the customer.

§ 9 PAYMENT

The payment to be made by the customer is in any event due at the time delivery of the software is effected. The due time for payment arises without any further precondition.

The payment to be made by the customer is to be transferred as stated on the invoice to the banking account of Paessler without deduction and free of expenses and costs for Paessler.

In the event of delay in payment the customer will pay to Paessler - without prejudice to compensation for further losses — the costs of judicial and extra judicial means and proceedings as well as interest at the rate of 5 % over the base interest rate of the European Central Bank.

§ 10 LIMITATIONS FOR CUSTOMERS IN THE UNITED STATES AND UNITED STATES TERRITORY

1. LIMITED WARRANTY

Paessler warrants that for a period of ninety (90) days from the date of download the software files will be free of defects as regards the product under normal use. Except for the foregoing, software is provided "AS IS". The customer's exclusive remedy and Paessler's entire liability under this limited warranty will be at Paessler's option to replace software media or refund the fee paid for the software. Any implied warranties on the software are limited to 90 days.

2. DISCLAIMER OF WARRANTY

UNLESS SPECIFIED IN THESE GENERAL TERMS AND CONDITIONS, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT ALLOWED BY APPLICABLE LAW.

3. LIMITATION OF LIABILITY

IN NO EVENT WILL Paessler BE LIABLE FOR ANY LOST OF REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF Paessler HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, EXCEPT TO THE EXTENT ALLOWED BY APPLICABLE LAW.

In no event shall Paessler's liability to the customer, whether in contract, tort (including negligence), or otherwise, exceed the price paid by the customer. The foregoing limitations shall apply even if the above-stated warranty fails its essential purpose.

If the Terms above are not applicable to the customer the following Limitations apply.

§ 11 INTERNATIONAL LIMITATIONS

1. Conformity of the Software

The software does not conform with these General Terms and Conditions if at the time the risk passes it is clearly different to the specifications, or in the absence of specifications, the software is not fit for the usual purpose.

Paessler is particularly not liable for the software being fit for a particular purpose to which the customer intends to put it or for the software's compliance with the legal requirements existing outside the Federal Republic of Germany.

2. Examination and Notice of Lack of Conformity

The customer must examine the software as required by law.

The customer shall give notice of any lack of conformity to Paessler as required by law, in any event directly and in writing and by the quickest possible means by which delivery is guaranteed (e.g. by telefax).

3. Consequence of Delivering non-conforming Software

Following due notice of lack of conformity, the customer can rely on the remedies provided for by the UN Sales Convention in regard to the terms laid down in these General Terms and Conditions. In the event of notice not having been properly given, the customer may only rely on the remedies if Paessler has fraudulently concealed the lack of conformity.

The customer is entitled to demand delivery of substitute software or repair or reduction of the purchase price as set forth in and in accordance with the terms of the UN Sales Convention.

Irrespective of the customer's remedies, Paessler is entitled to repair non-conforming software or to supply substitute software.

4. Third Party Claims and Product Liability

A. Third Party Claims

Without prejudice of further legal requirements, third parties' rights or claims founded on industrial or other intellectual property only found a defect in title to the extent that the industrial and intellectual property is registered and made public in the Federal Republic of Germany.

The customer's claims for defects in title including those founded on industrial or intellectual property will be time-barred according to the same rules as the claims for delivery of non-conforming software.

Third parties not involved in the conclusion of contract based on these General Terms and Conditions in particular those purchasing from the customer, are not entitled to rely on any remedy provided for in this General Terms and Conditions or to raise claims against Paessler, founded on delivery of non-conforming software or defect in title.

B. Product Liability

Without prejudice to Paessler's continuing legal rights and waiving any defense of limitation the customer will indemnify Paessler without limit against any and all claims of third parties which are brought against Paessler on the ground of product liability, to the extent that the claim is based on circumstances which are caused after risk passed by the customer.

5. Damages

A. Obligation to Pay Damages

Paessler is only obliged to pay damages pursuant to these General Terms and Conditions if it deliberately or in circumstances amounting to gross negligence breaches obligations owed to the customer. This limitation does not apply if Paessler commits a fundamental breach of its obligations.

Without prejudice to its continuing legal rights, Paessler is not liable for a failure to perform any of its obligations if the failure is due to impediments which occur, e.g. as a consequence of natural or political events, acts of state, industrial disputes, sabotage, accidents, or similar circumstances and which can not be controlled by Paessler through reasonable means.

The customer is required in the first instance to rely on other remedies and can only claim damages in the event of a continuing deficiency.

B. Amount of Damages

In the event of contractual or extra contractual liability Paessler will compensate the loss of the customer to the extent that it was foreseeable to Paessler at the time of the formation of the General Terms and Conditions.

§ 12 MISCELLANEOUS

1. Set off, Suspending Performance

Legal Rights of the customer to set-off against claims of Paessler for payment are excluded, except where the corresponding claim of the customer has either been finally judicially determined or recognized by Paessler in writing.

2. Place of Performance

The place of performance and payment for all obligations arising from the legal relationship between Paessler and the customer is Fürth (Germany).

3. Applicable Law

The legal relationship with the customer is governed by the United Nations Convention of 11 April 1980 on General Terms and Conditions for the International Sale of Goods (UN Sales Convention) in the English version.

Outside the application of the UN Sales Convention, the contractual and non-contractual legal relationship between the parties is governed by the non-uniform German law, namely by the BGB/HGB (German civil and commercial code).

4. Jurisdiction

The parties submit for all contractual and extra-contractual disputes arising from these General Terms and Conditions to the local and international exclusive jurisdiction of the courts having jurisdiction for Fürth (Germany).

5. Communication

All communications, declarations, notices, etc. are to be drawn up exclusively in the German or English language. Communications by means of e-mail or fax need to fulfill the requirement of being in writing.

6. Severability

If provisions of these General Terms and Conditions should be or become partly or wholly void, the remaining conditions continue to apply.

Please contact sales@paessler.com for information on other available license models and volume discounts.

